UNIVERSIDAD CÉSAR VALLEJO

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Stacking Ensemble Machine Learning usando métodos de balanceo de datos para la predicción de enfermedades de la columna vertebral

TESIS PARA OBTENER EL TÍTULO PROFESIONAL DE:

Ingeniero de Sistemas

AUTOR:

Gutierrez Allende, Diego Edú (orcid.org/0000-0001-8037-1570)

ASESOR:

Dr. Daza Vergaray, Alfredo (orcid.org/0000-0002-2259-1070)

LÍNEA DE INVESTIGACIÓN:

Sistemas de Información y Comunicaciones

LÍNEA DE RESPONSABILIDAD SOCIAL UNIVERSITARIA:

Promoción de la salud, nutrición y salud alimentaria

LIMA – PERÚ

2023

Dedicatoria

Se lo dedico a quienes me inspiraron, a quienes me ayudaron a llegar donde he llegado y por haberme forjado como la persona que soy en la actualidad, a mis padres.

Agradecimiento

Quiero agradecer a la universidad por abrirme las puertas brindarme la oportunidad de avanzar en mi carrera profesional. Agradezco especialmente a mi escuela profesional por su constante apoyo. Su fe en mis habilidades y su disposición para ayudarme han sido fundamentales para la finalización de esta tesis.

UNIVERSIDAD CÉSAR VALLEJO

FACULTAD DE INGENIERÍA Y ARQUITECTURA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Declaratoria de Autenticidad del Asesor

Yo, DAZA VERGARAY ALFREDO, docente de la FACULTAD DE INGENIERÍA Y ARQUITECTURA de la escuela profesional de INGENIERÍA DE SISTEMAS de la UNIVERSIDAD CÉSAR VALLEJO SAC - LIMA NORTE, asesor de Tesis Completa titulada: "Stacking Ensemble Machine Learning usando métodos de balanceo de datos para la predicción de enfermedades de la columna vertebral", cuyo autor es GUTIERREZ ALLENDE DIEGO EDÚ, constato que la investigación tiene un índice de similitud de 19.00%, verificable en el reporte de originalidad del programa Turnitin, el cual ha sido realizado sin filtros, ni exclusiones.

He revisado dicho reporte y concluyo que cada una de las coincidencias detectadas no constituyen plagio. A mi leal saber y entender la Tesis Completa cumple con todas las normas para el uso de citas y referencias establecidas por la Universidad César Vallejo.

En tal sentido, asumo la responsabilidad que corresponda ante cualquier falsedad, ocultamiento u omisión tanto de los documentos como de información aportada, por lo cual me someto a lo dispuesto en las normas académicas vigentes de la Universidad César Vallejo.

LIMA, 26 de Diciembre del 2023

Apellidos y Nombres del Asesor:	**Firma**
DAZA VERGARAY ALFREDO **DNI:** 40466240 **ORCID:** 0000-0002-2259-1070	Firmado electrónicamente por: ADAZAVE el 26-12-2023 12:25:04

Código documento Trilce: TRI - 0708310

INVESTIGA UCV

FACULTAD DE INGENIERÍA Y ARQUITECTURA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Declaratoria de Originalidad del Autor

Yo, GUTIERREZ ALLENDE DIEGO EDÚ estudiante de la FACULTAD DE INGENIERÍA Y ARQUITECTURA de la escuela profesional de INGENIERÍA DE SISTEMAS de la UNIVERSIDAD CÉSAR VALLEJO SAC - LIMA NORTE, declaro bajo juramento que todos los datos e información que acompañan la Tesis titulada: "Stacking Ensemble Machine Learning usando métodos de balanceo de datos para la predicción de enfermedades de la columna vertebral", es de mi autoría, por lo tanto, declaro que la Tesis:

1. No ha sido plagiada ni total, ni parcialmente.
2. He mencionado todas las fuentes empleadas, identificando correctamente toda citatextual o de paráfrasis proveniente de otras fuentes.
3. No ha sido publicada, ni presentada anteriormente para la obtención de otro gradoacadémico o título profesional.
4. Los datos presentados en los resultados no han sido falseados, ni duplicados, nicopiados.

En tal sentido asumo la responsabilidad que corresponda ante cualquier falsedad, ocultamiento u omisión tanto de los documentos como de la información aportada, por lo cual me someto a lo dispuesto en las normas académicas vigentes de la Universidad César Vallejo.

Nombres y Apellidos	**Firma**
DIEGO EDÚ GUTIERREZ ALLENDE **DNI:** 75426990 **ORCID:** 0000-0001-8037-1570	Firmado electrónicamente por: DEGUTIERREZG el 26-12-2023 19:20:46

Código documento Trilce: TRI - 0708313

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS Y FIGURAS

RESUMEN

El presente trabajo tiene como objetivo general, aplicar un sistema inteligente con ensemble machine learning con el método stacking mediante el uso de balanceo de datos que permitirá predecir enfermedades de la columna vertebral. Para ello, se realizó un estudio de enfoque cuantitativo y de tipo aplicada, con una población de 310 personas con enfermedades de la columna vertebral (Hernia, espondilolistesis), además tuvo un muestreo no probabilístico. El desarrollo del sistema se da mediante el uso de la metodología KDD, en la primera fase de selección de datos, se seleccionaron las variables relevantes para la predicción de enfermedades de la columna vertebral. En la segunda fase de preprocesamiento y tercera fase de transformación de los datos, se usaron diversas técnicas para tener listos los datos e iniciar con el entrenamiento del modelo en la cuarta fase de minería de datos, dónde se entrenaron doce algoritmos de Machine Learning (DT, RF, SVM, GNB, NC, NT, NTC, K-NN, GB, XGBoost, RL, AdaBoost). Finalmente, en la fase de evaluación, se evaluó el rendimiento del modelo con las métricas de exactitud, precisión, sensibilidad, F1-score, AUC, especificidad, MCC y la función los específicamente para los algoritmos NT y NTC. Los resultados mostraron que los algoritmos AdaBoost, RF, XGBoost, SVM y NB obtuvieron resultados con un índice mayor a 80% en todas las métricas. Se concluye que el sistema desarrollado es eficaz en la predicción de enfermedades de la columna vertebral.

Palabras claves: sistema inteligente, machine learning, stacking, selección de variables, predicción, enfermedades de la columna vertebral

ABSTRACT

The general objective of this work is to apply an intelligent system with ensemble machine learning with the stacking method through the use of data balancing that will allow predicting spinal column diseases. To this end, a study with a quantitative and applied approach was carried out, with a population of 310 people with spinal column diseases (Hernia, spondylolisthesis), and it also had a non-probabilistic sample. The development of the system occurs through the use of the KDD methodology, in the first phase of data selection, the relevant variables for the prediction of spine diseases are selected. In the second phase of preprocessing and third phase of data transformation, various techniques were used to have the data ready and begin training the model in the fourth phase of data mining, where twelve Machine Learning (DT) algorithms were trained). RF, SVM, GNB, NC, NT, NTC, K-NN, GB, XGBoost, RL, AdaBoost). Finally, in the evaluation phase, the performance of the model was evaluated with the metrics of accuracy, precision, sensitivity, F1-score, AUC, specificity, MCC and the function specifically for the NT and NTC algorithms. The results showed that the AdaBoost, RF, XGBoost, SVM and NB algorithms obtained results with a rate greater than 80% in all metrics. It is concluded that the developed system is effective in predicting spine diseases.

Keywords: intelligent system, machine learning, stacking, variable selection, prediction, spinal diseases

I. INTRODUCCIÓN

Ante la presencia de distintos problemas y enfermedades de columna vertebral que existen en la actualidad, se ha dificultado la predicción rápida del diagnóstico sobre que trastorno de la columna vertebral puede tener el paciente. La columna vertebral puede padecer de distintas patologías principales degenerativas, como lo son las hernias discales lumbares, estenosis vertebral, tumores vertebrales, hernias discales cervicales, escoliosis, discopatía lumbar degenerativa, cervicoartrosis, fracturas vertebrales, entre otras.

Según Noriega-Álvarez et al. (2021) mencionan que, los problemas en la columna vertebral humana siguen siendo frecuentes en nuestra sociedad y entre los problemas más frecuentes están incluidos el dolor lumbar, deformidades de la columna vertebral, tumores y lesiones vertebrales como traumatismos, infecciones, etc. Se hace mención a la patología degenerativa de la columna vertebral, siendo una de las principales causas importantes de dolor y malestar que genera alteraciones en un gran porcentaje de adultos, con una prevalencia de 60 a 90% y teniendo mayor afección en la quinta década de la vida humana. Por otro lado, para poder conocer los distintos tipos de enfermedades de la columna vertebral, según (Randolph et al., n.d.) menciona que las enfermedades de la columna vertebral se pueden presentar en distintas edades, algunas patologías de la columna vertebral en los niños y adultos es; la escoliosis idiopática, siendo una deformidad de la columna vertebral mayor de 10º teniendo una causa desconocida (el 22% de los pacientes con curvatura escolióticas de 20º tienen alguna alteración del eje neutral y el 80% requerirán tratamiento); la escoliosis congénita, la cual no tiene un patrón de herencia especifico y tiene su aparición de forma aislada, la cifosis, los tipos de cifosis más comunes son la postural, la de Scheuermann (deformidad estructural de la columna toráxica)y la congénita; la espondilólisis/espondilolistesis es una enfermedad que puede tener distintas causas; el síndrome de Klippel-Feil que se caracteriza por ser un desperfecto de segmentación de la columna vertebral cervical; la calcificación de los disco intervertebrales es una de las patologías más frecuentes de la columna cervical; la patología de la apófisis odontoides que se desarrolla a partir de la osificación de dos centros, fusionándose antes de los tres meses de edad.

En el ámbito internacional, los autores (Cui et al., 2022) mencionan que, las enfermedades de la columna vertebral se encuentran entre las causas más comunes de dolor y discapacidad en todo el mundo. Teniendo un lugar importante las imágenes ya que representan un importante procedimiento de diagnóstico de cuidado de la columna. Las investigaciones de imágenes pueden proporcionar información y conocimientos que no son visibles a través de una inspección visual ordinaria o humana. El aumento de la utilización de esta inspección humana ha creado múltiples desafíos para un departamento de radiología o practica derivada, e incluso también en otras disciplinas como la predicción del resultado del tratamiento.

En el ámbito nacional (Gómez, C., 2021) menciona este grupo de patologías abarca entre el 60 a 90% de población que tuvieron un dolor lumbar en su vida, siendo una de las causas principales del 15% de incapacidades laborales en el mundo, teniendo un índice de personas menores de 45 años. Por otro lado (Infanzon y Riveros, 2022) mencionan que, las alteraciones de la columna torácica se generan por un mecanismo compensatorio, se empieza cuando la personase encuentra sentada, tendrá la columna dorsal flexionada, por lo tanto, la columna torácica se flexionara y al estar un largo tiempo en esa posición, el cuerpo se acostumbra a ese hábito y los músculos se irán debilitando, generando dolor torácico y va acompañado con dolor muscular.

Los trastornos de la columna vertebral (McGrath et al., 2022), se refieren a una gama amplia de anomalías que ocurren a lo largo de la vida de una persona en respuesta a la carga sobre la columna vertebral y también el factor genético. La degeneración de la columna es el proceso normal de envejecimiento y es patológica si existen síntomas asociados. Los trastornos pueden surgir por el uso excesivo y el trauma crónico. En la columna cervical y lumbar, específicamente C5-C7 Y L4-S1, son las partes más comunes de cambios degenerativos debido a factores de estrés biomecánicos.

Para poder conocer más trastornos de la columna, es necesario centrarse en los distintos trastornos pediátricos de la columna que son más numerosos y diferentes a comparación de la población adulta, según (Bachmann, 2021) durante

los periodos de rápido crecimiento, la columna vertebral en desarrollo puede experimentar alteraciones en los patrones de crecimiento.

Se plantea como solución la implementación de un sistema inteligente de machine learning con el método stacking ensemble y balanceo de datos para la predicción de enfermedades de la columna vertebral. La propuesta de solución, primeramente, se tomarán datos de personas con enfermedades de la columna vertebral, luego se preprocesarán los datos de la población y así poder obtener una muestra, se procederá a la fase de investigación y esta fase consta de tres partes, primeramente, se procesarán los datos, luego se procederá al modelamiento estadístico mediante el uso de los distintos algoritmos y para finalizar esta fase se obtendrá la predicción. Por lo tanto, se generará la propuesta de solución y así tener la predicción del tipo de enfermedad de la columna vertebral que pueda tener el paciente evaluado.

Las causas principales de la columna vertebral según (González, et. al, 2000), tienen distintos factores, entre algunos factores se encuentran el factor genético, hereditario, malas posturas, la edad, siendo el dolor de espalda uno de los motivos más frecuentes por los cuales el paciente acude al médico y existe una preocupación constante en la temprana edad que se presentan estos problemas. Los principales motivos del dolor son los desequilibrios causados por un inadecuado método en la adopción de posturas; pero se excluyendo los traumatismos, patologías, malformaciones y todas aquellas alteraciones de la columna vertebral.

Las consecuencias generadas tienen una gran variedad, dependiendo de la gravedad y de cuál sea la causa de la enfermedad, según Sjeklocha y Gatz (2021) el tipo de enfermedad de la columna vertebral que tenga el paciente generara distintos tipos de complicaciones, las más frecuentes son infecciones, tumores recurrentes, dolor, debilidad motora y así habrá una gran variedad de consecuencias según la gravedad y el tipo de patología que sufra la persona.

Por ello se plantea el problema general ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir enfermedades de la columna vertebral?, como problemas

específicos, para poder medir los datos se plantean las siguientes preguntas ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la exactitud (Accuracy)?, ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la precisión?, ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la exhaustividad (Recall)?, ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la F1-score?,¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función del área bajo la curva ROC (Area Under the ROC Curve – AUC-ROC)?, ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la especificidad (Specificity)?, ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de MCC (Matthews Correlation Coefficient)? y como ultimo problema específico se plantea la siguiente pregunta ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la pérdida (Loss)?

La presente investigación se justifica en cuatro ámbitos, porque a través del proyecto se obtiene un sistema inteligente que usara la tecnología de machine learning mediante el método stacking ensemble para la predicción de enfermedades de la columna vertebral. Asimismo, se presenta la justificación en el ámbito tecnológico porque la implementación de un sistema inteligente con machine learning stacking ensemble y balanceo de datos permitirá predecir el tipo y la gravedad de la enfermedad de la columna vertebral, logrando así poder obtener un diagnóstico y su debido tratamiento.

En el ámbito teórico la investigación se justifica porque permite generar conocimientos de los fundamentos de la predicción con machine learning, el método stacking ensemble, el balanceo de datos y las enfermedades de la columna vertebral. De esa manera se facilita su comprensión y aplicación en sistemas inteligentes con stacking ensemble machine learning, generándose conocimiento valioso para poder usarse con otras finalidades que permita ayudar en distintas enfermedades. En el ámbito social se justifica porque los beneficiarios con este sistema inteligente serán los médicos y pacientes, que les permitirá obtener un diagnóstico y así poder generar un tratamiento hacia el paciente. Y se justifica en el ámbito practico porque el presente estudio ayudara a los médicos en la predicción de enfermedades de la columna vertebral y así poder generar un rápido diagnostico para el paciente.

La investigación tiene como objetivo general, aplicar un sistema inteligente con ensemble machine learning con el método stacking mediante el uso de balanceo de datos que permitirá predecir enfermedades de la columna vertebral. Además, haciendo referencia a los problemas específicos, se plantean los siguientes objetivos específicos, aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la exactitud (Accuracy), aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la precisión (Precision), aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la exhaustividad, aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la F1-score, aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función del área bajo la curva (Area Under the ROC Curve – AUC-ROC), aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la especificidad, aplicar un sistema inteligente con stacking ensemble

machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de MCC (Matthews Correlation Coefficient) y como ultimo objetivo específico aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la perdida (Loss).

En la investigación, para poder demostrar el objetivo general se planteó como hipótesis general que, el sistema inteligente con ensemble machine learning con el método stacking mediante el uso de balanceo de datos ayuda y mejora significativamente predecir las enfermedades de la columna vertebral. Además se plantearon las siguientes hipótesis específicas, el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la exactitud (Accuracy), el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la precisión, el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la exhaustividad (Recall), el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la F1-score, el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función del área bajo la curva (Area Under the ROC Curve – AUC-ROC), el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la especificidad, el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función MCC (Matthews Correlation Coefficient) y como última hipótesis específica el sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la perdida (Loss).

II. MARCO TEÓRICO

En este apartado, se ha realizado una indagación exhaustiva de fuentes informativas, considerando antecedentes tanto a nivel nacional como internacional, e integrando teorías y fundamentos conceptuales pertinentes. A continuación, se ofrece un desglose completo de los términos y conceptos fundamentales que respaldarán el progreso de la investigación, ofreciendo una perspectiva clara sobre cómo serán utilizados en este capítulo y en el contenido global de la investigación.

Tabla Nº 1: Diccionario de términos

DT	Árbol de decisión o Decision Tree
GB	Impulso de Gradiente o Gradient Boosting
KNN	K-vecinos más cercanos o K-Nearest Neighbors
NB	Bayes ingenuo o Naive Bayes
NC	Centroide más cercano o Nearest Centroid
RF	Bosque Aleatorio o Random Forest
RN	Redes Neuronales o Neural Network
RNC	Redes Neuronales Convolucionales o Convolutional Neural Network
RL	Regresión Logística o Logistic Regression
SVM	Máquina de Vectores de Soporte o Support Vector Machine
XGB	Potenciación Extrema del Gradiente o Extreme Gradient Boosting
ADA	Potenciación adaptativa o Adaptive Boosting
ML	Aprendizaje automatico o Machine Learning
KDD	Knowledge Discovery in Databases o descubrimiento de conocimiento en bases de datos

Fuente: Elaboración propia

La columna vertebral es de vital importancia para el desarrollo rutinario de la vida humana en distintas actividades motrices. En la presente investigación se realizará la implementación de machine learning usando métodos de balanceo de datos y la técnica stacking ensemble, para ello es necesario tener antecedentes para poder tener un buen desarrollo de la investigación.

En la investigación del autor (Martinez G., 2022), titulada “Desarrollo de un clasificador basado en redes neuronales para la detección de escoliosis en imagen rx de columna”. El objetivo principal de este estudio es entrenar un clasificador basado en el algoritmo de redes neuronales convolucionales con una base de datos de imágenes RX de escoliosis. Se obtuvo como resultados que el algoritmo de redes neuronales convolucionales tuvo dificultades en la clasificación cuando se realizó la clasificación de 2 clases (escoliosis tipo C y tipo S) pero tuvo un porcentaje de clasificación de 79,2%. Luego procedió a agregar otra clase de pacientes sanos y el rendimiento mejoró, obteniendo un 94% de exactitud o Acuraccy. Se concluyo que la validación del algoritmo fue exitosa y fue realizada mediante el uso de la métrica de exactitud y la matriz de confusión, en donde usaron imágenes de RX de pacientes sanos y con escoliosis del “Hospital Sagrado Corazón de Jesús”. El algoritmo CNN pudo clasificar a los pacientes con escoliosis de tipo C, en donde se obtuvo un 95% de Acuraccy, para la clasificación de pacientes con escoliosis de tipo S se obtuvo un 97.3% de Acuraccy y para pacientes sanos obtuvo un 95% de Acuraccy.

Por otro lado, en la investigación de (Zhang et al., 2023) ,titulado “Automated machine learning-based model for the prediction of delirium in patients after surgery for degenerative spinal disease”. El objetivo de estudio de esta investigación es usar algoritmos de aprendizaje automático (XGBoost, Logistic, Random Forest, Adaboost, KNN, SVM, GaussianNB, multilayer perceptrón, extreme gradient bosting) para identificar variables críticas y predecir el delirio postoperatorio (POD) en pacientes con enfermedad degenerativa de la columna. El estudio fue de carácter experimental. Se obtuvo como resultados que, el modelo XBOSST superó y funciono bien ante los otros modelos clasificadores en el entrenamiento conjunto, en donde se obtuvo los resultados (métrica área bajo la curva [AUC]: 92,8%, intervalo de confianza [IC] del 95%: 90,1% -95,0%), y en el conjunto de validación

se obtuvo los siguientes resultados AUC: 87%, IC del 95%: 80,7% -93,3%. Se usaron doce variables, las cuales son: edad, albumina, hipertensión, presión sanguínea mínima intraoperatoria, enfermedad pulmonar, pérdida de sangre intraoperatoria, medida de confianza, enfermedad cardiovascular-cerebrovascular, intervalo de tiempo entre la admisión y la cirugía y como variable final se usó la diferencia máxima de presión sanguínea entre la admisión y el intraoperatorio, donde el modelo XBOSST obtuvo una precisión de 85,71%. Se concluyo que un modelo de machine learning y un predictor web para el POD de la enfermedad espinal degenerativa se desarrolló con éxito para demostrar el grado de riesgo de POD durante el periodo perioperatorio y así tener medidas preventivas para pacientes de alto riesgo.

En el artículo de investigación realizado por (Varçın et al. 2021), titulado “End-To-End Computerized Diagnosis of Spondylolisthesis Using Only Lumbar X-rays”. Tuvo como obetivo desarrollar un modelo de CNN (Convolutional Neural Netowork) basado en transfer learning que solo usa radiografías lumbares. El modelo se entrenó con 1922 imágenes, de las cuales 187 se utilizaron para la validación. Posteriormente, se evaluó el rendimiento del modelo utilizando un conjunto adicional de 598 imágenes. En el proceso de entrenamiento, se emplea Yolov3 para extraer las regiones de interés (ROIs). Se procedió a dividir las ROIs en conjuntos de entrenamiento y validación, luego fueron aplicadas a MovilNet CNN ajustada finamente para acabar con el entrenamiento. Se obtuvo como resultados que el modelo CNN alcanzo una precisión de prueba del 99%, una sensibilidad de la prueba del 98% y la especificidad de la prueba del 99%. Se concluyo que el modelo CNN funcionó correctamente, teniendo buenos resultados de rendimiento y se resalta que el modelo puede ser utilizado en clínicas externas donde no haya expertos presentes.

En la investigación de (Ansari et al., 2013), titulado “Diagnosis of Vertebral Column Disorders Using Machine Learning Classifiers”. El objetivo de estudio de esta investigación es proponer el diagnóstico y clasificación de trastornos de la columna vertebral utilizando clasificadores de aprendizaje automático, incluidas redes neuronales de retropropagación de alimentación directa, redes neuronales de regresión generalizada y máquinas de soporte vectorial, y evaluar su

rendimiento. El conjunto de datos que se utilizó en esta investigación fue recopilado, mediante el uso de imágenes de resonancia magnética (IRM) y se clasifica en tres clases diferentes: hernia de disco, espondilolistesis y normal. Los resultados experimentales demostraron que la red neuronal de retropropagación de alimentación directa tiene una precisión del 93.87% en casos de prueba desconocidos y funciona mejor que los otros métodos.

En la investigación de (Castro et al., 2020), titulado “Early detection of ankylosing spondylitis using texture features and statistical machine learning, and deep learning, with some patient ageanalysis”. En donde se propuso el primer uso de clasificadores basados en aprendizaje automático estadístico y aprendizaje profundo para detectar la erosión, un síntoma temprano de la espondilitis anquilosante, mediante el análisis de imágenes de tomografía computarizada (TC), en donde se tomó en cuenta la edad del paciente en proceso. Se usaron matrices de co-ocurrencia, de niveles de gris y patrones binarios locales para generar características de entrada para algoritmos de ML, específicamente para KNN y Random Forest. Se obtuvo como resultados que el modelo de Random Forest superó a los modelos de KNN, en donde fueron evaluados con las métricas de precisión promedio de validación cruzada, recall y área bajo la curva ROC, en donde obtuvieron los valores de 96.0%, 92.9% y 0.97, respectivamente, para erosión vs. pacientes de control jóvenes, y 82.4%, 80.6%, y 0.91, respectivamente, para erosión vs. pacientes de control ancianos. En base a los resultados se concluyó que el clasificador de aprendizaje profundo entrenado sin minimizar la pérdida de validación fue el mejor y logra una precisión, recall y AUC ROC de validación cruzada de ocho pliegues del 99.0%, 97.5% y 0.97, respectivamente, para erosión vs. todos (combinados jóvenes y ancianos) los pacientes de control. Este clasificador superó a un radiólogo musculoesquelético con 9 años de experiencia en sensibilidad y especificidad cruda en un 8.4% y 9.5%, respectivamente. Como conclusión final se da a conocer que los resultados indican el potencial del aprendizaje automático y profundo para ayudar en el diagnóstico de la espondilitis anquilosante.

En la investigación de (Chen et al., 2018) , titulado “Design of a Clinical Decision Support System for Fracture Prediction Using Imbalanced Dataset”. Tiene

como objetivos dar a conocer la asociación de los corticosteroides inhalados y la fractura y diseñar un sistema de apoyo clínico de predicción de fracturas. En la metodología, se usaron datos de pacientes de más de 20 años, que haya visitado centros de salud en su pasado, específicamente entre los años 2002 y 2010, donde les hayan recetado corticosteroides, estos datos se recuperaron de “The National Health Insurance Research Database (NHIRD)”. Se tomaron los criterios de inclusión y exclusión, donde se excluyeron a los pacientes diagnosticados con fracturas de cadera o fracturas de vertebras antes de usar corticosteroide. La población total de este estudio fue de 11645 pacientes que recibieron tratamiento con corticosteroides inhalados y entre esa población 1134 pacientes fueron diagnosticados con fractura de cadera o fractura vertebral. Se obtuvieron como resultados que las enfermedades respiratorias crónicas y las variables relacionadas con los corticosteroides. El desarrollo del sistema se desarrolló con un algoritmo integrado y el algoritmo de máquina de vectores de soporte, donde se entrenó y valido los datos, donde se pudo obtener datos balanceados mediante los métodos de random oversampling. Los resultados que se obtuvieron fueron que el rendimiento del sistema presento una sensibilidad (recall) del 69,84 al 77,00% y un área bajo la curva de 0,7495 a 0,7590. La conclusión de esta investigación es que a largo plazo los corticosteroides inhalados pueden incluir osteoporosis y una mayor incidencia de fracturas de cadera o de vertebras. Debe de haber un monitoreo continuo en personas de mayor de edad y mujeres después de la menopausia.

En la investigación de (Karabacak y Margetis, 2023), titulado “Machine Learning-Based Prediction of Short-Term Adverse Postoperative Outcomes in Cervical Disc Arthroplasty Patients”. Esta investigación tuvo como objetivo evaluar la efectividad de los algoritmos de aprendizaje automático (ML) en predecir resultados adversos a corto plazo después de la artroplastia discal cervical (ADC) y crear un aplicativo web fácil y accesible. Usaron 4 algoritmos (XGBoost, LightGBM, CatBoost y Random Forest) de ML para desarrollar modelos predictivos, y estos modelos se incorporaron a una aplicación web de acceso abierto. Para evaluar los modelos usaron las métricas de AUC ROC y precisión, en donde obtuvieron un 0.814 y 87.8% para todos los algoritmos. Se concluyo que los enfoques de ML tienen el potencial de predecir resultados postoperatorios después de la cirugía de ADC. A medida que la cantidad de datos en cirugía espinal

aumenta, el desarrollo de modelos predictivos como herramientas de toma de decisiones, son clínicamente útiles puede mejorar significativamente la evaluación de riesgos y el pronóstico.

En la investigación de los autores (Unal Y., Polat K. y Erdinc H., 2014), titulado “Pairwise FCM based feature weighting for improved classification of vertebral column disorders”. Esta investigación tuvo como objetivo proponer un método de preprocesamiento de datos para mejorar el rendimiento de clasificación y determinar automáticamente trastornos de la columna vertebral, incluyendo hernia discal (DH), espondilolistesis (SL) y normal (NO). Se propuso un método de ponderación de características basado en pairwise fuzzy C-means (FCM), se crearon los siguientes grupos (DH-SL, DH-NO y SL-NO) y se ponderaron utilizando un conjunto de características basado en FCM. Se usaron los algoritmos de perceptrón multicapa (MLP), k-vecinos más cercanos (k-NN), Naive Bayes y máquina de vectores de soporte (SVM). Para evaluar el rendimiento de los modelos se usaron las métricas de precisión de clasificación, sensibilidad, especificidad, curvas ROC y medida F1-score. Se obtuvo como resultados que los valores de medida F1-score obtenidos fueron 0.7738 para el clasificador MLP, 0.7021 para k-NN, 0.7263 para Naive Bayes y 0.7298 para el clasificador SVM en la clasificación del conjunto de datos de trastornos de la columna vertebral con tres clases. Con el método de ponderación de características basado en pairwise fuzzy C-means, los valores de medida F1-score obtenidos fueron 0.9509 para MLP, 0.9313 para k-NN, 0.9603 para Naive Bayes y 0.9468 para el clasificador SVM. Se concluyo que el método propuesto de ponderación de características basado en pairwise fuzzy C-means es robusto y efectivo en la clasificación del conjunto de datos de trastornos de la columna vertebral. El mejor rendimiento para la clasificación del conjunto de datos se obtuvo con el clasificador Naive Bayes.

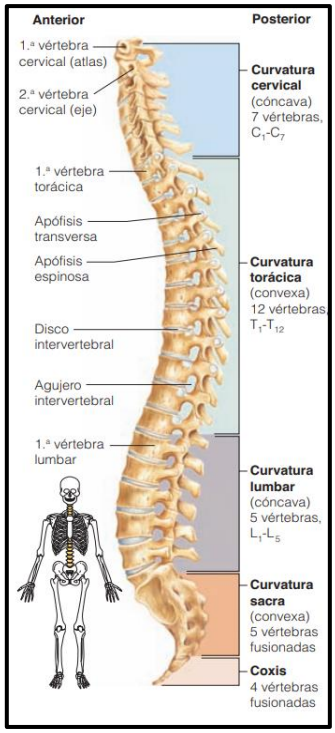
Por otra parte, se hace uso de las siguientes teorías y bases conceptuales, para que sirvan como guía durante todo el desarrollo. Por otra parte, poder conocer más a fondo sobre las enfermedades de la columna vertebral, se escogieron: la espondilolistesis que proviene de la espondilitis y las hernias discales.

Para poder saber que es la espondilolistesis, es necesario conocer sobre la espondilitis anquilosante, en base a ello el autor (Zhu et al., 2019) menciona que

la espondilitis anquilosante es una enfermedad autoinmune, que afecta principalmente a las articulaciones de la columna, las articulaciones sacroilíacas que son las que unen la pelvis con la columna vertebral y los tejidos blandos adyacentes; existen casos más avanzados donde esta inflamación generará una fibrosis y calcificación de los huesos de la columna vertebral, dando como resultado la perdida de flexibilidad y como consecuencia más grave la fusión de la columna. Por otro lado, los autores (Van Der Linden et al., 2022) los rasgos característicos de la espondilitis anquilosante son: dolor vertebral inflamatorio crónico, dolor torácico, dolor de glúteos alternante, uveítis anterior aguda, sinovitis), entesitis sacroilitis radiográfica, antecedentes familiares de espondilitis anquilosante, enfermedad intestinal inflamatoria crónica y soriasis.

Luego de haber conocido la descripción de la espondilitis anquilosante, se citó al autor (Lan et al.,2023), para conocer que es la espondilolistesis, en donde menciona que es el desplazamiento relativo de las vértebras superiores e inferiores debido a diversas razones, la cual tiene una incidencia de 6% de la población en general, con una proporción de dos hombres por cada mujer.

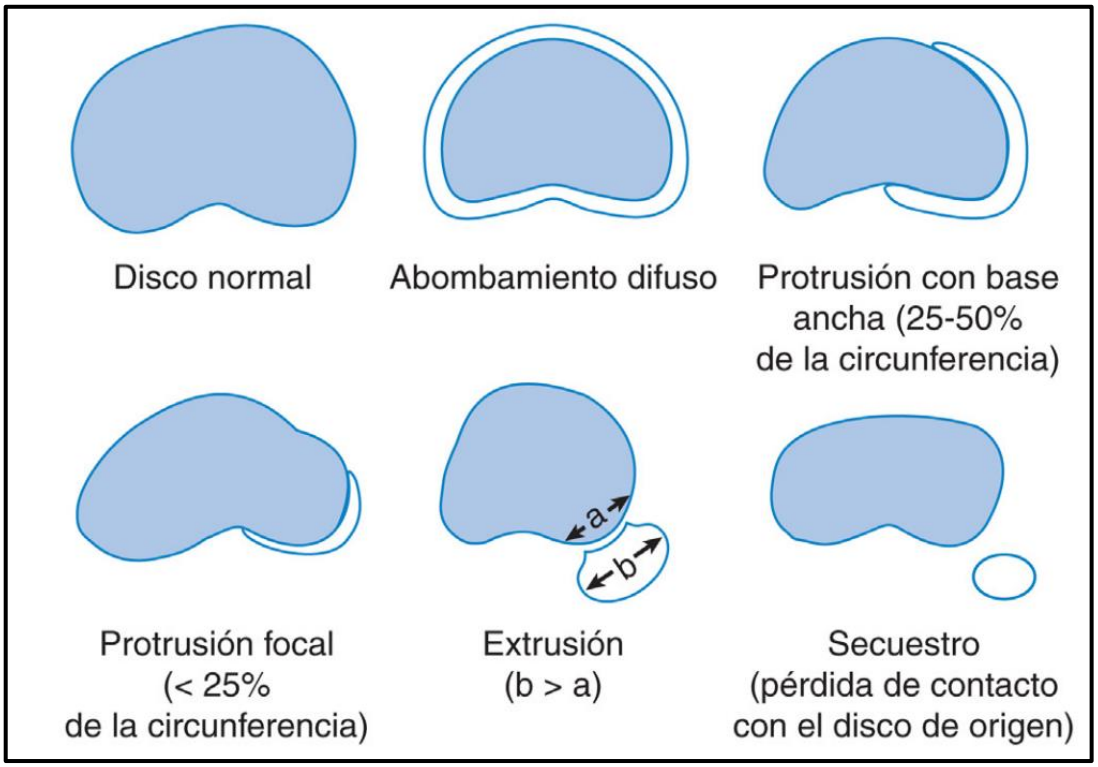
Figura Nº 1: Parte anterior y posterior de la columna vertebral y sus subdivisiones



Fuente: (Anatomía y Fisiología Humana, 2014, p. 176)

Las hernias discales, (Barr et al., 2022) menciona que el material discal desplazado puede clasificarse inicialmente como abombamiento, eso significa desplazamiento de más del 25% de la circunferencia discal, o como hernia, que afecta al 25% de la circunferencia. Estas se pueden clasificar en protrusiones o extrusiones. Las hernias discales pueden describirse como contenidas o no contenidas según la integridad de las fibras externas del anillo. Más del 95% de las hernias discales lumbares se localizan en los niveles L4-L5 Y L5-S1, seguidos por L3-L4 y L2-L3. Las hernias discales pueden causar una respuesta inflamatoria que puede afectar a la raíz nerviosa o comprensión mecánica y ambas pueden generar síntomas radiculares. Por otro lado (Rajiv D., 2022.) menciona que, la hernia discal es la causa más frecuente de dolor radicular en adultos jóvenes, sostiene que el 85% de los casos de dolor radicular se asocian al disco intervertebral herniado.

Figura Nº 2: Hernia, protrusión y extrusión discal



Fuente: (Trastornos lumbares, 2022)

Según (Fulkerson, n.d.) afirma que el nivel con más frecuencia afectado es L4-L5, el patrón de hernia en adultos se observa luego de la osificación fisaria completa. Las hernias graves pueden causar radiculopatía o síndrome de causa equina.

Antes de conocer la técnica que será implementada, es necesario conocer más sobre la tecnología de machine learning. Según (Charles et al., 2023) los diversos métodos de aprendizaje automático (ML) son un subtipo de inteligencia artificial, se originan en la informática y usan algoritmos establecidos, en base al análisis de una base de datos para la realización de tareas .Por otro lado, según (Liang et al., 2022) menciona que el machine learning (ML) se puede usar para avanzar rápidamente en el tratamiento de la columna, incluso diferentes tecnologías de ML han sido usadas para clasificar curvaturas de la columna y facilitar el diagnóstico precoz. Asimismo (Pineda, 2022) afirma que, el machine learning son técnicas computacionales modernas que permiten llegar a mejores resultados, sino que cada vez existe un uso más frecuente en la práctica clínica con el procesamiento en tiempo real, de la gran cantidad de datos que se vaya a manejar y así va a permitir predecir cada vez más datos y obtener una mayor exactitud distintas situaciones de salud en distintas patologías y enfermedades existentes, pudiendo tener una intervención rápida; de forma inmediata para el área de emergencias y de forma preventiva para distintas posibles enfermedades.

Respecto a los métodos que pueden aplicarse para facilitar el uso del ML, se encuentra **la selección de variables o características**, donde el autor Gualdrón (2006) menciona que la selección de variables abarca un conjunto de técnicas de reducción de dimensionalidad de los datos en el procesamiento, el cual tiene como objetivo encontrar un subconjunto óptimo de variables que minimice la perdida de información, también menciona que en todo sistema tiene variables que pueden contribuir o no al correcto funcionamiento del sistema. Algunas variables son de gran utilidad, pero otras no (p.82). Por este motivo se busca seleccionar las variables con ayuda de filtros que permitan el correcto funcionamiento de la selección de variables.

Para ello se usó primeramente el algoritmo **SelectkBest**, en donde Sabab et al. (2021), mencionan que la técnica de selección de características Select K-Best es univariante en naturaleza. Utiliza distintas pruebas estadísticas univariadas y selecciona K-Best funciones del conjunto de funciones

En cuanto al algoritmo RFE, los autores Tao et al. (2024) mencionan que el algoritmo **RFE** (eliminación de características recursivas), puede extraer

características útiles de los datos de origen y es una de las técnicas más utilizadas de extracción de variables, la cual consiste en seleccionar un evaluador del modelo para determinar la importancia de las variables (p.3).

Para conocer sobre el algoritmo **SelectPercentile**, los autores Birfir, Elalouf y Rosenbloom (2021), mencionan que el algoritmo SelectPercentile y SelectkBest son similares, con la diferencia que permite que el usuario especifique un porcentaje determinado de funciones (en lugar de un número determinado) según sus puntuaciones.

Y para finalizar se usó el algoritmo de **PCA**, los autores Hayati et al. (2024) mencionan que es una técnica de reducción de datos que crea componentes principales (PC), que son combinaciones lineales de las variables originales, y crean variables nuevas y no correlacionadas (p. 4).

Respecto al algoritmo de optimización de hiperparámetros se usó **GridSearchCV**, los autores Sihab et al. (2022) mencionan que si se desglosa el termino GridSearchCV, se obtiene GridSearch y CV, en donde todos los hiperparámetros se seleccionan en función sobre el cual GridSearch optimization (GSO) construye el espacio de GridSearch, también mencionan que la función GridSearchCV es usado para ajustar automáticamente los hiperparámetros para determinar los valores de parámetros óptimos.

La técnica que se usara en análisis de datos y el machine learning es el balanceo de datos o resampling, para poder abarcar y solucionar el desequilibrio en la distribución de clases en un conjunto de datos. Según García, J (2021), menciona que las técnicas hibridas de undersampling y oversampling pueden combinarse en el balanceo de datos. Lo más frecuente es aplicar distintos métodos, combinando una técnica de eliminación de instancias redundantes en la clase mayoritaria y luego aplicar la técnica de oversampling. Por otro lado, según Yuan et al. (2023) afirma que, el balanceo de datos mejora significativamente el rendimiento de los modelos, especialmente en el score-F1. Según Cruz-Ruiz et al. (2023) menciona que el uso de balanceo de datos puede mejorar el rendimiento de los modelos de conjuntos de datos de clases desequilibradas. Según Bae et al. (2021) menciona

que el balanceo de datos se puede realizar con algoritmos como SMOTE, ROSE Y ADASYN.

Para poder conocer más sobre undersampling se hace referencia a (Mohammed et al., 2020), donde menciona que el submuestreo es el proceso de disminuir la cantidad de instancias o muestras mayoritarias, algunos métodos comunes contienen enlaces de tomeks, centroides de conglomerados y otros métodos, puede ser realizado aumentando la cantidad de instancias o muestras de clase minoritaria con la producción de nuevas instancias o la repetición de alguna instancia. Por otro lado (Lööv, 2020), afirma queel oversampling es un método para tratar datos equilibrados, su propósito es equilibrar los datos filtrando algunos datos para tener el mismo número de ejemplos para cada clase, es un método rápido cuando se usan datos desequilibrados debido a que se ignora muchos ejemplos de clase mayoritaria.

Los algoritmos que se usaran para poder aplicar undersampling son Random Undersampling y tomek-link.

Para comenzar a conocer sobre el primer algoritmo de undersampling. El autor Bach M. (2022), menciona que la ventaja de la técnica undersampling es la reducción del tiempo de entrenamiento, lo que es importante en el conjunto de datos muy desequilibrados con un gran número de instancias de clases mayoritarias. El Random undersampling trata de equilibrar la distribución mediante la eliminación aleatoria de ejemplos de clases mayoritarias, su principal inconveniente del Random undersampling es que puede eliminar datos que son muy importantes para el machine learning y para ello a lo largo de los años se propusieron estrategias heurísticas de undersampling, entre una de las categorías de los métodos heurísticos de undersampling se basa en el algoritmo de K-vecinos. Por otro lado (Leevy et al., 2023) según menciona que el uso de undersampling se da para que los conjuntos de datos estén lo suficientemente equilibrados.

Por otro lado, el segundo algoritmo Tomek-link, los autores (AT et al., 2016) mencionan que es un método de undersampling desarrollado por Tomek. Según (Pereira et al., 2020) afirmó que el método tomek link puede usarse como una técnica de undersampling o como un paso de limpieza posterior del proceso, si es

usada como una técnica de undersampling solo se eliminan las muestras de la clase mayoritaria.

El oversampling es una técnica utilizada para abordar el desequilibrio de clases en conjunto de datos, para ello el desequilibrio de clases ocurre cuando una clase esta subrepresentada en comparación con las demás clases. Según (Zheng et al., 2015) el oversampling aumenta el número de instancias de clases minoritarias para equilibrar la distribución de clases, el tipo de oversampling más común es el oversampling aleatorio, que simplemente duplica las posiciones minoritarias, una de las debilidades principales es que no presenta información nueva al conjunto de datos y esto puede generar un ajuste excesivo.

Los algoritmos que se usaran para poder aplicar oversampling son:

Para comenzar a conocer sobre el primer algoritmo de oversampling. Los autores Xu et al. (2023) mencionan que la función principal del algoritmo de Random oversampling (ROS) es superar el problema del desequilibrio redistribuyendo el conjunto de datos de entrenamiento. La función principal del algoritmo ROS es extraer aleatoriamente muestras de las clases minoritarias y hacer múltiples repeticiones para equilibrar la distribución de clases del conjunto de entrenamiento

Por otro lado, el algoritmo SMOTE, Según los autores Li et al. (2023) el método de sobremuestreo SMOTE sintetiza ejemplos de la clase rara, específicamente, primero elige una observación de la clase rara y selecciona aleatoriamente un ejemplo de sus k vecinos más cercanos.

Para continuar con la realización de los modelos stacking. El stacking está compuesto de dos etapas principales: la etapa de entrenamiento y la etapa de predicción. En la etapa de entrenamiento, se utilizan diferentes algoritmos de aprendizaje para construir varios modelos base utilizando conjuntos de datos de entrenamiento. Los algoritmos que serán utilizados en la etapa de entrenamiento del stacking son:

Para ello se utilizó como primer algoritmo, decisión tree, en donde el autor Ramírez, C. (2020) mencionó que el árbol de decisión consiste en la formulación de un numero de preguntas condicionales y así para cada pregunta, se dividan en

datos hasta conseguir claramente a que categoría pertenecen. Por otro lado (Nebot, 2018), menciona que el árbol de decisiones puede mostrar valores estadísticos inferenciales y además permite la inclusión de variables cuantitativas y cualitativas.

Continuando con el segundo algoritmo **Gradient Boosting** (GB), los autores Bai y Chandra (2023) mencionan que es una técnica de aprendizaje en conjunto que se puede aplicar para tareas de clasificación y regresión. El modelo de Gradien Boosting es construido por etapas de forma similar a otros métodos de boosting, pero se pueden generalizar a cualquier función de pérdida diferenciable, haciéndolas muy flexible (p.3).

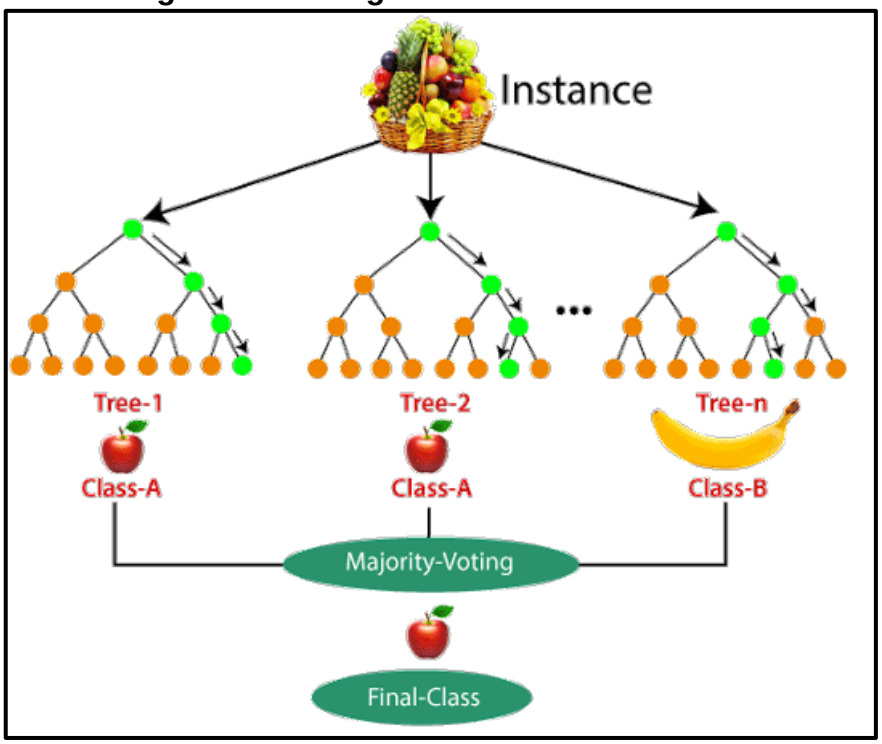
Para conocer más sobre el tercer algoritmo K-Nearest Neighbors o K-vecinos más cercanos (KNN), la empresa tecnológica multinacional estadounidense IBM, afirma que **KNN** es un clasificador de aprendizaje supervisado no paramétrico que usa la proximidad para hacer predicciones sobre la agrupación de un punto de datos individual y generalmente es usado en problemas de clasificación.

En cuanto al cuarto algoritmo **Naive Bayes** el autor Alarcón C. (2015) menciona que el algoritmo también es llamado clasificador bayesiano simple, su función es asignar a un objeto en la clase con mayor probabilidad y supone que todas las variables o atributos son condicionalmente independientes dado el valor de la clase.

En cuanto al quinto algoritmo Nearest Centroid Classifier (**NCC**), los autores Hari, Abdurrahman y Afif (2020) menciona que es un método basado en centroides y su principal ventaja es que no requiere parámetros, por lo que los resultados no son determinados por la configuración de parámetros, pero únicamente por la distancia entre los datos. El método NCC se utiliza para determinar el centroide más cercano al punto de datos.

Respectivamente al sexto algoritmo, según (He et al., 2022) define que **random forest** está basado principalmente y tiene como idea base, el uso de decisión tree y aprendizaje automático de conjuntos de Bagging. Este tipo de algoritmo da como resultado de salida el promedio de los resultados del árbol de decisión múltiple o también llamado moda y así puede permitir a ayudar la precisión del árbol de decisión y se puede obtener una mejor aplicación del modelo.

Figura Nº 3: Diagrama de Random Forest



Fuente: (Sruthi E., 2023)

En función al séptimo algoritmo se hace referencia a los autores (Kim Soon et al. (2020), que mencionan que las **redes neuronales** se encuentran entre uno de los algoritmos más populares en el machine learning, que ha tenido un buen rendimiento en muchos dominios. Por otro lado, según (Naumetc, 2016) el tipo más simple de red neuronal es una red de perceptores de una sola capa, que consiste en una sola capa de nodos de salida y sobre las redes neuronales complejas, que las neuronas en capas ocultas pueden representar dificultades que los humanos no pueden entender.

Continuando con el octavo algoritmo, el autor Bonilla (2020) menciona que las redes neuronales convolucionales (**CNN**) está compuesta por múltiples capas para calcular la salida de un conjunto de datos. El desarrollo de este algoritmo comenzó en la década de 1950 y la capa más principal es la capa convolucional.

Respectivamente con el noveno algoritmo, los autores González et al. (2017), afirman que las **SVM** son entrenadas por algoritmos de optimización convexa y generadas de una estructura que depende de un subconjunto de vectores de soporte. Por otro lado, según afirman que (G. Cano et al., 2017) son un

conjunto de métodos de aprendizaje automático supervisado que se pueden aplicar a una variedad de problemas de clasificación o regresión. Originalmente se usaron para clasificar clases de objetos linealmente separables. Las SVM se han utilizado en varios campos, como la química en el diseño de fármacos.

Respectivamente al décimo algoritmo, según los autores Chen y Guestrin (2016) mencionan que el algoritmo Extreme Gradient Boosting (**XGBoost**) surgió como una implementación eficiente del método Gradient Boosting. Este algoritmo tiene como principales características, el incluir términos de penalización para evitar el sobreajuste. El factor primordial de XGBoost es su escalabilidad en todos los escenarios. La escalabilidad de XGBoost se debe a varios sistemas importantes y optimizaciones algorítmicas.

Continuando con el onceavo algoritmo **AdaBoost**, los autores Freund y Schapire (1997), quienes fueron los desarrolladores de este algoritmo, mencionan que este algoritmo consiste en crear varios predictores sencillos en secuencia, para que de forma consecutiva generen ajustes correctos que no se realizaron en el primer ajuste, en el tercer ajuste un poco mejor que el segundo ajuste no pudo ajustar y así consecutivamente.

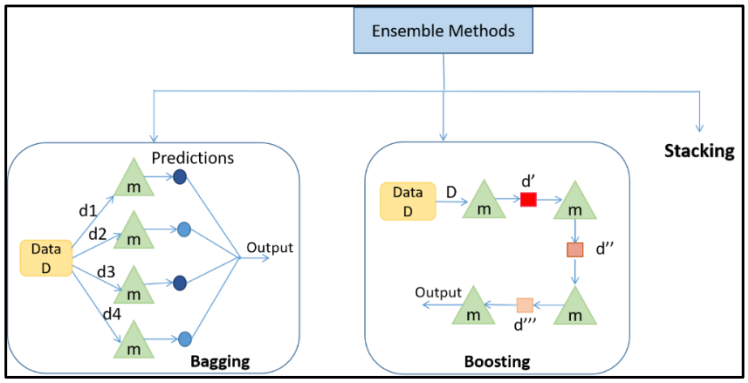
En la etapa de predicción, se utilizan los modelos base entrenados para hacer predicciones sobre nuevos datos. Luego, las predicciones de los modelos base se combinan utilizando un modelo de nivel superior llamado "meta-modelo". El algoritmo comúnmente usado como meta-modelos en el stacking es:

Finalmente, se planteó como doceavo algoritmo el uso del algoritmo de **regresión logística** que servirá como meta-modelo en la composición de los modelos stacking. Según Irene Moral Peláez (n.d.) menciona que, se selecciona el mejor modelo de regresión logística comparando modelos utilizando la razón de verosimilitud. Es un modelo estadístico que se usa para predecir una variable binaria.

Luego de haber implementado los modelos de algoritmos se hizo uso del método Stacking Ensemble, que permitirá generar conjuntos de algoritmos que fueron mencionados previamente. En base a ello, según (Ma et al., 2018) indica que es un método de conjunto que utiliza un modelo superior para combinar

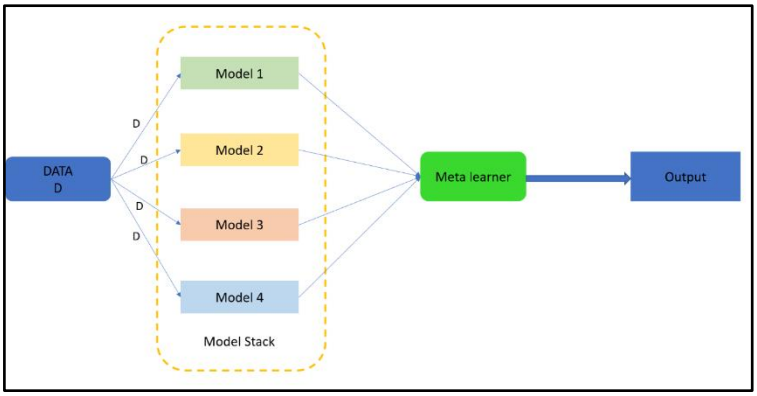
modelos de nivel inferior para lograr obtener una mayor precisión de predicción. El modelo stacking ensemble presenta superioridad en la mejora de las predicciones (Yin et al., 2023). Por otro lado (Satapathy et al., 2021) menciona que es un método de ensamblaje, que se encuentra en la arquitectura de dos capas, los modelos y datos de la primera capa se consideran datos y los modelos de la capa base. La principal ventaja del modelo stacking ensemble permite mejorar la predicción, también ayuda a identificar continuamente las muestras de predicción errónea de las características de los clasificadores de la capa base.

Figura Nº 4: Representación de los tipos de métodos de ensemble



Fuente: Kandelwal, Y (2021)

Figura Nº 5: Algoritmo de apilamiento



Fuente: Dou et al. (2020)

Para evaluar y comparar el rendimiento de un modelo stacking ensemble, se pueden utilizar varias métricas para medir su desempeño. Asimismo, dentro de las Métricas de evaluación del modelo, se definen las métricas de **exactitud**,

precisión, **sensibilidad**, **F1-score**, **AUC**, **especificidad**, **MCC** y la función de **pérdida**, que serán evaluadas a través de una matriz de confusión.

La autora Chamat (2021) lo define como una métrica de desempeño para evaluar el rendimiento del modelo predictivo, la matriz de confusión “Confusion Matrix” de la librería de Sklearn, la cual va permitir evaluar la precisión de la predicción en el proceso de clasificación binaria.

Tabla Nº 2: Matriz de confusión

		Estimación por el modelo	
		Negativo	**Positivo**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

Fuente: Jiménez y Merino (2022)

Dónde:

- True Negatives (TN): Predicciones negativas que son realmente negativas.

- False Positives (FP): Predicciones positivas que son realmente negativas.

- False Negatives (FN): Predicciones negativas que son realmente positivas.

- True Positives (TP): Predicciones positivas que son realmente positivas.

Para iniciar con la medición del modelo se usará la métrica de exactitud, ya que se usará para elegir el mejor modelo optimo. Según los autores Borja-Robalino et al. (2020) es la métrica más usada por los investigadores para evaluar el rendimiento un modelo de aprendizaje automático, debido a se puede calcular fácilmente y así comprender para poder evaluar la efectividad del algoritmo, su desventaja principal es que produce menos valores discriminatorios y distintivos para el caso multiclase desbalanceado, la fórmula es:

$$\frac{TP + TN}{TP + TN + FP´FN}$$

Según (Aqil et al., 2022) ;los valores True Negative (TN) son datos que están correctamente clasificadas como salidas negativas o falsas. Verdadero Positivo (TP) es datos que se clasifican correctamente como resultados positivos o verdaderos. Falso positivo (FP) son datos que se clasifican incorrectamente si la salida es positiva o verdadera. Falso Negativo (FN) son datos que son clasificado incorrectamente.

Adicionalmente para evaluar el rendimiento del modelo en función a las predicciones realizadas. El autor Ramos Ponce (2020), menciona que es un valor que se indica cuantas instancias de todas las que fueron elegidas como positivas, son realmente positivas o la probabilidad de que sean positivas, por otro lado (Villegas Cubas & Niño, n.d.) menciona que es la proporción de instancias clasificadas correctamente como positivas, su fórmula es:

$$Precisión = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

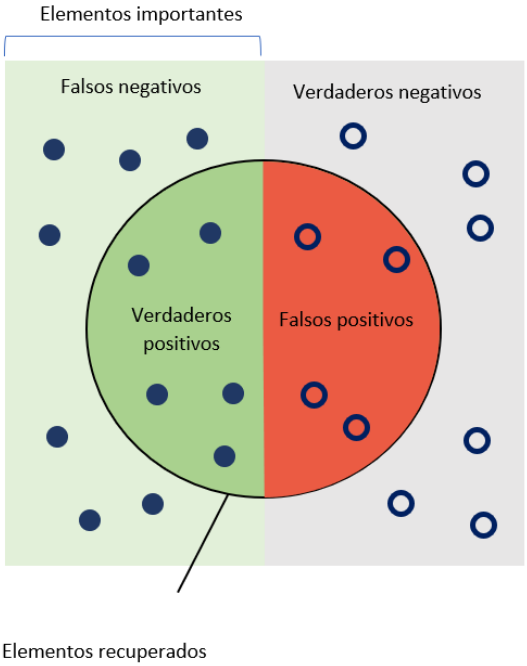
		Predicción	
		0	1
Realidad	0	TN	FP
	1	FN	TP

Según (Mendoza Olguín et al., 2019) el recall se encarga de comprobar que porcentaje de los elementos significativos para un usuario le fueron efectivamente recomendados. Por otro lado (Menasalvas et al., n.d.) menciona que el recall o exhaustividad es la fracción de ejemplos clasificados correctamente del total de ejemplos elegidos de una determinada clase. Su fórmula es:

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Por otro lado, para la métrica F1-score, el autor Dalianis, H. (2018) menciona que la puntuación F1 significa la media armónica entre precisión y recuperación según la función de ponderación, significa la media armónica entre la precisión y recuperación, la puntuación F1 puede tener diferentes índices que otorgan distintos pasos en la precisión y recuperación. Según (C. Cano & Ruiz, 2018) también es llamado F-score o F-measure, la fórmula es:

$$\mathrm{F1} = 2.\frac{precission\,.\,recall}{precission + recall}$$



Para poder hablar sobre el área bajo la curva es necesario mencionar la curva ROC, que es una herramienta estadística que se usa para evaluar la capacidad discriminativa de una prueba diagnóstica dicotómica. Según (Martínez Pérez & Pérez Martin, 2023) menciona que dentro de la curva ROC existe un área llamada área bajo la curva, que se define como la probabilidad de clasificar correctamente a un par de individuos seleccionados al azar.

$$AUC=\int_{0}^{1} ROC(x)\,dx$$

Para evaluar la capacidad de los modelos e identificar los negativos verdaderos se usará la métrica de especificidad. Según (Bravo & Cruz, 2015) menciona que la métrica es usada en clasificación para poder evaluar la capacidad de un modelo y así poder identificar la clase negativa. Es la proporción de verdaderos negativos que fueron identificados correctamente, del total de individuos. Por lo tanto, se puede decir que la especificidad es el cociente entre los

verdaderos negativos y falsos positivos, esas proporciones son parámetros inherentes a la prueba diagnóstica menos dependientes. Según (Dymaxion Labs, 2021) la fórmula de la especificidad es:

$$Especificidad = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

Continuando con las métricas, los autores Chicco & Jurman (2020) afirman que MCC fue desarrollado por Matthews en 1975. Por otro lado, según (Liu et al., 2015) el Matthews Correlation Coefficient, abreviado como MCC es esencialmente el coeficiente de correlación entre lo observado y predicho en la clasificación binaria. Los autores Chicco, Tötsch, et al. (2021) mencionan que MCC es una métrica robusta que resume el desempeño del clasificador en un solo valor, si son positivos o negativos tienen igual importancia. Finalmente, los autores Chicco, Starovoitov, et al. (2021) afirman que el coeficiente de Matthews es más informativo y confiable que la precisión, score-f1, precisión equilibrada y otros más.

$$MCC = \frac{(\mathrm{TP} * \mathrm{TN} - \mathrm{FP} * \mathrm{FN})}{\sqrt{((\mathrm{TP} + \mathrm{FP}) * (\mathrm{TP} + \mathrm{FN}) * (\mathrm{TN} + \mathrm{FP}) * (\mathrm{TN} + \mathrm{FN}))}}$$

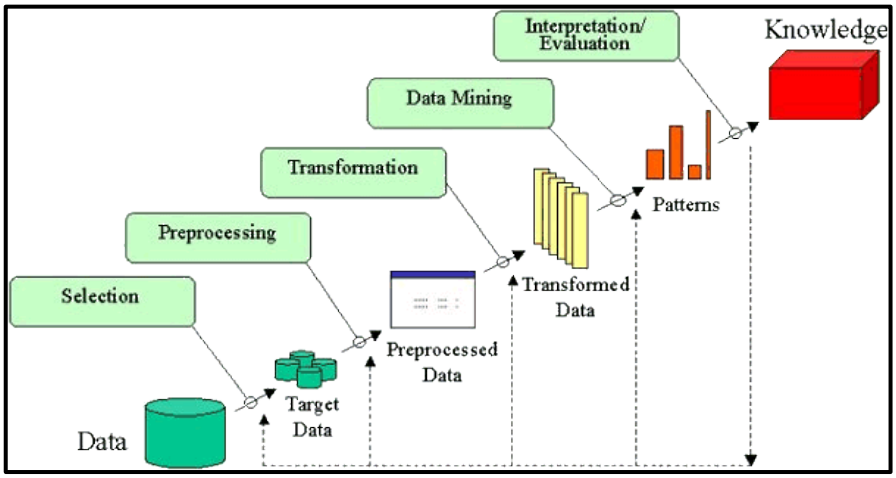
Finalmente, la función de pérdida (Loss) que es mayormente usada en RN y es aplicada durante el entrenamiento, que proporcionará información sobre la discrepancia entre las predicciones del modelo y los valores verdaderos. Según (Ciampiconi et al., 2023) la función de pérdida (Loss) depende del problema a resolver, los datos disponibles y el tipo de algoritmo de machine learning, se presentaron 33 funciones de perdidas, siendo complicado poder encontrar la función de perdida apropiada, divide los tipos de perdida en; perdida de regresión, perdidas de clasificación, perdidas generativas y perdidas de base energética, cada una de las perdidas con distintos tipos perdidas dentro según su función. Por otro lado (Wang et al., 2022) menciona que la función de perdida es importante en la construcción de algoritmos de machine learning y su mejora de rendimiento. Existen muchas funciones de perdida por ello se propuso dividirlas en funciones de perdida en el machine learning tradicional y en el aprendizaje profundo. Se presentaron 21 funciones de perdida, donde los de machine learning tradicional incluye 11 funciones de perdida para problemas de clasificación, 6 funciones de perdida para

problemas de regresión y 4 para aprendizaje no supervisado. La fórmula para problemas de clasificación es la entropía cruzada categórica. Su fórmula es la siguiente:

$$\text{Cross Entropy} = -1/n * \Sigma(\Sigma(yij * \log(\hat{y}ij)))$$

Continuando con la metodología que se va aplicar, se hace referencia a la metodología KDD, que va a permitir tener un enfoque estructurado y sistemático para obtener información valiosa, para ello según (Saghari et al., 2023) afirma que en el año 1996 se introdujo el concepto de **KDD** para extraer el conocimiento de una base de datos que va a permitir reducir el nivel de complejidad y las incertidumbres causadas por la falta de conocimiento. Por otro lado (Romei et al., 2006), afirma que la metodología KNOWLEDGE DISCOVERY IN DATABASES (KDD) ha obtenido mayor valor en lo que se refiere a diseño de algoritmos eficientes de extracción de conocimiento. Según Nwagu et al. (n.d.) existen distintos conceptos sobre KDD y menciona que su función principal es extraer conocimiento de datos de nivel inferior. Se recalca que para poder lograr este método es necesario la interacción humana.

Figura Nº 6: Diagrama de la metodología KDD



Fuente: (Saghari et al., 2023)

El software **anaconda** se usará para la gestión de los paquetes, el desarrollo y análisis del sistema inteligente. Según (Rolon-Mérette et al., 2020) es un software gratuito que provee herramientas necesarias para la investigación y la ciencia. Este software permite codificar en Phyton, son conocidos también como entornos de

desarrollo integrado (IDE). Este tipo de entornos contienen distintas características útiles para escribir, editar y depurar código, visualizar e inspeccionar datos, almacenar variables y colaborar en proyectos. Anaconda es una interfaz gráfica de usuario simple, que incluye bibliotecas más importantes y tiene una fácil instalación; permite simplificar la actualización de todas las bibliotecas usadas y asi poder usar Phyton con distintos IDE, bibliotecas y funciones.

También se usará el entorno de **Jupyter** y se hará uso de sus distintas funciones, para ello, según Rolon et al. (2020) mencionan que para poder usar Phyton se podrá ejecutar a través de Jupyter Notebook, el es de código abierto y permite crear y ejecutar código en distintas disciplinas, incluida la simulación numérica y el aprendizaje automático.

Para poder realizar la programación se hará uso del lenguaje de **Python**, para poder conocer más sobre el lenguaje, en el libro “Introduccion a la programación con Phyton”, los autores Marzal Varó y Gracia Luengo (2009) mencionan que es un lenguaje de programación con algunas ventajas que lo hace más interesante y didáctico, sus características principales son: es un lenguaje expresivo, programas compactos; Python es muy legible, un lenguaje elegante y permite una lectura fácil; un entorno interactivo; el entorno de Python detecta muchos errores de programación, posee una variedad de estructuras fáciles de manipulas y entre otras características.

Por otra parte, Grabner (2020) menciona que **Spyder** es un entorno de desarrollo integrado - Integrated Development Environment (IDE). Esto le permite combinar muchas características asociadas con tareas de programación, que incluyen: la edición de archivos de texto, la ejecución de código en un Shell, la vista previa de figuras creadas por su código y mucho más.

III. METODOLOGÍA

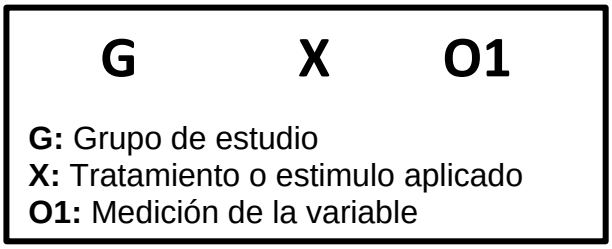
En este capítulo, se profundiza en el enfoque y la estructura de la investigación, incluyendo elementos esenciales como la clarificación y definición de las variables. También se establecen los límites de la población, se detalla el procedimiento para seleccionar la muestra y se proporciona un análisis exhaustivo de las técnicas e instrumentos empleados para recopilar datos, así como del método utilizado para analizarlos. Se abordan también consideraciones éticas relacionadas con el trabajo de investigación.

3.1. Tipo y diseño de investigación

Esta investigación es de tipo aplicada, debido a que se aplica el conocimiento de machine learning mediante la técnica stacking ensemble que combina múltiples modelos para mejorar la predicción y la técnica de balanceo de datos para abordar desequilibrios en los conjuntos de datos. Y así se permitirá y generará la predicción de enfermedades de la columna, Según (Castro Maldonado et al., 2023) , la investigación aplicada son trabajos originales que están enfocados principalmente en la obtención de nuevos conocimientos y así poder dar solución a los problemas específicos, previamente identificados de un contexto específico. Se recurre a los conocimientos obtenidos y alcanzados durante la búsqueda de información en la investigación básica y así poder encaminarlos al cumplimiento de los objetivos específicos.

El diseño de investigación es experimental de tipo preexperimental, se utilizó datos de personas con enfermedades de la columna vertebral y posteriormente se aplicó machine learning haciendo uso de las técnicas stacking ensemble y balanceo de datos, para obtener una predicción. Según Ramos-Galarza (2021), este es un subdiseño de un estudio experimental en el que la variable independiente tiene un solo nivel, que es el grupo de estudio, y por lo tanto puede recibir una intervención impuesta por el investigador.

Figura Nº 7: Diagrama de diseño preexperimental



G: Personas con enfermedades de la columna vertebral (Hernia o espondilolistesis)

X: Sistema inteligente con métodos de balanceo de datos y stacking ensemble para predecir enfermedades de la columna vertebral.

O1: Métricas de evaluación de modelo

3.2. Variables y operacionalización

Las variables que se identificaron en el proyecto de investigación son: machine learning, variable independiente y la variable predecir enfermedades de la columna vertebral, variable dependiente; a su vez la variable dependiente cuenta con una dimensión de enfermedades de la columna vertebral, cuenta con ocho indicadores como exactitud, precisión, exhaustividad, F1-score, área bajo la curva, especificidad, MCC y pérdida; de tal manera que la operacionalización de variables se detalla en el Anexo Nº 02 Y 03.

3.2.1. Variable Independiente

La predicción de enfermedades de la columna vertebral, representa a la patología, dificultad o afección que afecta a la estructura y la función de la columna vertebral. Según (Gallucci et al., 2007), la enfermedad vertebral degenerativa ha sido estudiados por bioarqueología, y tienen un vínculo con el estilo de vida y el nivel de actividad física de las personas. La enfermedad degenerativa de la columna vertebral es una definición que abarca distintos espectros de anomalías degenerativas, afecta a las estructuras óseas y al disco intervertebral, aunque distintas patologías están relacionadas porque el factor patógeno es el principal identificado en la sobrecarga crónica, siendo así una de las causas principales del dolor de espalda, radiculopatía y discapacidad.

3.2.2. Variable Dependiente

Para predecir las enfermedades de la columna vertebral se ha tomado en cuenta una dimensión, enfermedades de la columna vertebral y ocho indicadores, (a) Exactitud, (b) Precisión, (c) exhaustividad, (d) F1-score, (e) Área bajo la curva, (f) especificidad, (g) MCC y la (h) perdida; la escala de medición que se estableció es la razón. La operacionalización a detalle de estas variables se encuentra en el ANEXO N° 2.

3.3. Población, muestra y muestreo

3.3.1. Población:

En la presente investigación no se enfoca en una población especifica, porque no está limitado a un grupo de personas o población definida. Ya que se seleccionó datos de una base de datos publica de personas con enfermedades de la columna vertebral. Según (Toledo et al., n.d.) la población de una investigación está compuesta por personas, objetos, organismos e historias clínicas, que previamente han sido definidas y delimitadas en el problema de investigación. Las características principales para delimitar la población son el contenido, lugar y tiempo.

Se realizó una búsqueda de bases de datos, se usó el repositorio de Kaggle como recolector de datos y se seleccionó la BD de VertebralColumnDataSet. Por lo tanto, la población está conformada por 310 personas con problemas de la columna vertebral, donde las enfermedades destacadas de esta base de datos son las hernias vertebrales y espondilolistesis.

Las variables con las que cuenta la base de datos son pelvic_incidence**,** pelvic_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius, degree_spondylolisthesis y class. Para poder realizar la explicación de las variables se realizó la Tabla Nº2, para conocer a profundidad el significado y el rango de las variables.

Tabla Nº 3: Descripción y rango de las variables

Variables	Descripción	Rango
pelvic_incidence	**Incidencia pélvica:** Es la medida del ángulo entre la línea perpendicular al plano de la pelvis en el punto de rotación del sacro y la línea que conecta este punto con la vértebra cervical.	**Valor Min.** 18.58 – 26.148 **Valor Max.** 129.834 – 195.32
pelvic_tilt	**Inclinación pélvica:** Representa el ángulo entre la línea que	**Valor Min.** -6.555, -9.84

	conecta el centro de la cabeza y el punto medio de la línea entre las dos espinas ilíacas anterosuperiores y la línea perpendicular al plano de la pelvis.	**Valor Max.** 49.432 – 74.26
lumbar_lordosis_angle	**Ángulo de lordosis lumbar:** Es el ángulo formado por las vértebras lumbares en la columna vertebral	**Valor Min.** 8.45 – 14 **Valor Max.** 125.742 - 212.31
sacral_slope	**Inclinación sacra:** Es el ángulo entre la base de la columna vertebral y la línea horizontal	**Valor Min**. 7.94 - 13.367 **Valor Max.** 121.43- 182.79
pelvic_radius	**Radio pélvico:** La distancia media entre el punto central de la cabeza del fémur y el centro de la línea que conecta las espinas ilíacas anterosuperiores	**Valor Min.** 65.86 – 70.083 **Valor Max.** 153.27 – 163.071
degree_spondylolisthesis	**Grado de espondilolistesis:** Es una medida de la cantidad de deslizamiento de una vértebra sobre la vértebra subyacente. Puede ser positivo o negativo dependiendo de la dirección del deslizamiento.	**Valor Min**. -0.84, -11.058 **Valor Max.** 418.543 - 5859
class	**Clase:** Esta variable está clasificada en 3 clases: Hernia, Normal y Espondilolistesis	0 (Hernia) 1 (Normal) 2 (Espondilolistesis)

3.3.2. Muestra:

Según (Otzen & Manterola, 2017) la representatividad de una muestra permite extrapolar y así generalizar los resultados. Una muestra será representativa depende de si se selecciona al azar, de modo que los sujetos de la población y los sujetos accesibles en la población tengan la misma probabilidad de ser seleccionados de la muestra y, por lo tanto, incluidos en el estudio; por otro lado, la muestra es el número de sujetos. es numéricamente representativa de la parte total de la población. Sin embargo, en este estudio se consideró que la muestra estará representada por 310 personas con enfermedades de la columna vertebral, en donde se ha considerado la muestra igual a la población ya que permitirá tener una mayor cantidad de datos para el desarrollo del experimento.

3.3.3. Muestreo:

El presente trabajo aplicó un muestreo no probabilístico porque existió la dificultad de obtener datos de pacientes que presenten alguna enfermedad de la columna vertebral y mediante el uso del muestreo no probabilístico se disminuyó el tiempo de búsqueda de datos. En la investigación de (Gómez, A. & Gómez, K., 2019) menciona que el muestreo no probabilístico es el más usado en las ciencias sociales, ya que necesita una estructura estadística que en varias ocasiones no es accesible a los investigadores.

3.3.4. Unidad de análisis:

En base a los mencionado, se tomó como unidad de análisis al paciente con enfermedad de la columna vertebral, ya que permite un enfoque más completo y centrado en el paciente.

3.4. Técnicas e instrumentos de recolección de datos

Para poder realizar la selección de la base de datos. Se realizo una búsqueda dentro de distintos repositorios como Kaggle (comunidad de científicos de datos y profesionales de machine learning centrada en la inteligencia artificial), UCI Machine Learning Repository, Microsoft DataSet y el buscador de Google de Datasets. Se realizó una revisión de datos sobre enfermedades de la columna vertebral y se hizo la selección de la BD de VertebralColumnDataSet. Para poder

cumplir con el control de calidad de este proyecto se cumple con la verificación y revisión previa de los datos, para obtener una muestra adecuada.

3.5. Procedimientos

En el presente trabajo, inició con la búsqueda de información e investigaciones similares al tema propuesto, tanto a nivel nacional como internacional, ya sea, de tesis o artículos científicos para poder conocer a profundidad las enfermedades más comunes que afecte la columna vertebral. Luego de haber realizo el proceso de búsqueda, se estableció los antecedentes y bases teóricas que apoyaron el desarrollo del proyecto y así poder establecer la dimensión e indicadores de la variable dependiente.

Para poder conseguir los datos, primero se realizó una selección de las distintas enfermedades de la columna vertebral y se seleccionaron la espondilolistesis y las hernias vertebrales. La selección de la BD se dio a través del cumplimiento de los objetivos planteados en la investigación. Se prosiguió con la verificación de datos, la conversión de datos de formato csv. a xlsx. Se valido que todos los datos fueran correctos y se realizó el previo análisis de datos para la selección de la muestra. Luego de haber recopilado la información, para ejecutar los algoritmos, se hizo uso de la herramienta Jupyter Notebook y Anaconda. Luego se entrenó el modelo con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA. Al finalizar el entrenamiento de los modelos, se realizó una comparación de los resultados de precisión de los algoritmos y se generó los modelos stacking en función a los mejores resultados de los algoritmos. Así se pudo determinar los mejores modelos de predicción. Posteriormente, se construyó la interfaz del Sistema inteligente. Dicha interfaz, estuvo desarrollada en Spyder con el lenguaje de programación Python y SQL Server.

3.6. Método de análisis de datos

Se realizó una búsqueda en distintos repositorios de datos en donde se seleccionó el repositorio de Kaggle y por lo tanto se prosiguió a buscar información vinculada con las enfermedades de la columna vertebral y se seleccionó la base de datos, llamada VertebralColumnDataSet, donde cuenta con datos sobre las enfermedades de la columna vertebral, específicamente la espondilitis y las hernias discales. La base de datos cuenta con una división de cada enfermedad

mencionada y así cuentan con las variables de incidencia pélvica, inclinación pélvica, ángulo de lordosis lumbar, pendiente sacra, radio pélvico, el grado de espondilolistesis y el tipo de clase. Se usarán los modelos de regresión logística y regresión lineal, para poder analizar los datos estadísticos.

Se realizo un análisis inferencial, para poder hacer inferencias y generalizaciones de la población seleccionada con enfermedades de la columna vertebral, haciendo referencia y basándose principalmente en la muestra de 310 pacientes. Se aplico este método para poder determinar las diferencias de probabilidad y predicción de hernias vertebrales, espondilolistesis y personas normales. Luego, se empleó las fichas de registro según las métricas de evaluación de modelo y los algoritmos seleccionados. Finalmente, se usó Pyhton, Jupyter Notebook y Anaconda. De esa manera se estableció el mejor modelo de predicción para el desarrollo del sistema inteligente.

3.7. Aspectos éticos

Los datos son públicos, por lo tanto, no se necesitó un consentimiento informado de los pacientes. Los métodos que se usaran brindan transparencia y honestidad para las pruebas de predicción que se realizarán y así poder obtener resultados y conclusiones transparentes. Para poder asegurar la integridad académica se seguirá el manual ISO para citar de manera correcta y en función a las fuentes de información usadas. Para poder cumplir con los permisos correspondientes y tomar las medidas necesarias, para obtener los permisos necesarios de la Universidad Cesar Vallejo y así poder proseguir con los lineamientos de la RESOLUCION DE CONSEJO UNIVERSITARIO N.° 0101-2022/UCV que se visualiza en el Anexo Nº18, con el propósito de poder cumplir con los requisitos de cumplimiento y así llevar a cabo de manera apropiada y cumplir con las reglas de esta investigación. Estos permisos aseguran que se respeten los datos adquiridos del repositorio Kaggle y cumplir con la debida confidencialidad y los requisitos éticos que se realicen en la investigación.

IV. RESULTADOS

En este apartado, se demuestran los resultados obtenidos de la investigación, basados en los indicadores de especificidad, precisión, sensibilidad, exactitud, área bajo la curva, MCC, F1-score y la perdida, que fueron comparados entre los doce algoritmos: Decision Tree, Random Forest, k-Nearest Neighbors, redes neuronales, redes neuronales convolucionales, XGBoost, Naive bayes, Nearest Centroid, SVM, AdaBoost, Gradient Boosting y regresión logística. En base a los algoritmos mencionados, se crearon 5 modelos stacking en función a la precisión que obtuvieron los modelos.

Tabla Nº 4: Composición de los Modelos de Stacking

Nº Stacking	Algoritmos
1	• XGBoost • Naive Bayes • Support Vector Machine • Random Forest • Logistic Regression (meta-modelo)
2	• k-Nearest Neighbors • Naive Bayes • Support Vector Machine • Random Forest • Logistic Regression (meta-modelo)
3	• XGBoost • k-Nearest Neighbors • Support Vector Machine • Random Forest • Logistic Regression (meta-modelo)
4	• XGBoost • Adaboost • Support Vector Machine • Random Forest • Logistic Regression (meta-modelo)
5	• XGBoost • Decision Tree • Support Vector Machine • Random Forest • Logistic Regression (meta-modelo)

Fuente: Elaboración propia

Se realizó la codificación de variables, ya que permitió convertir las clases en valores numéricos enteros para el correcto funcionamiento de los algoritmos.

Tabla Nº 5: Codificación de las variables

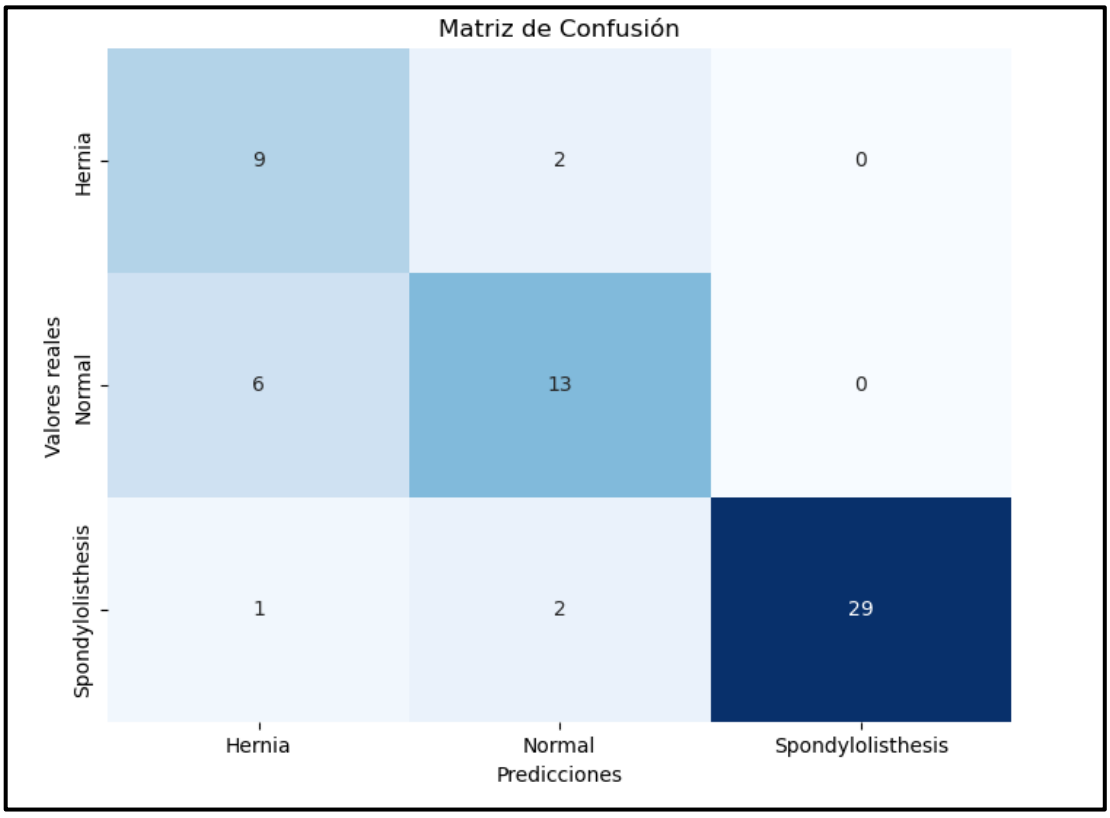
Variable: class	Valores
Hernia	0
Normal	1
Espondilolistesis	2

Fuente: elaboración propia

A continuación, se demuestra la validación de cada objetivo mediante la matriz de confusión.

- **Decision Tree**

Figura Nº 8: Matriz de Confusión – DT



Fuente: elaboración propia

Tabla Nº 6: Matriz de Observación - DT

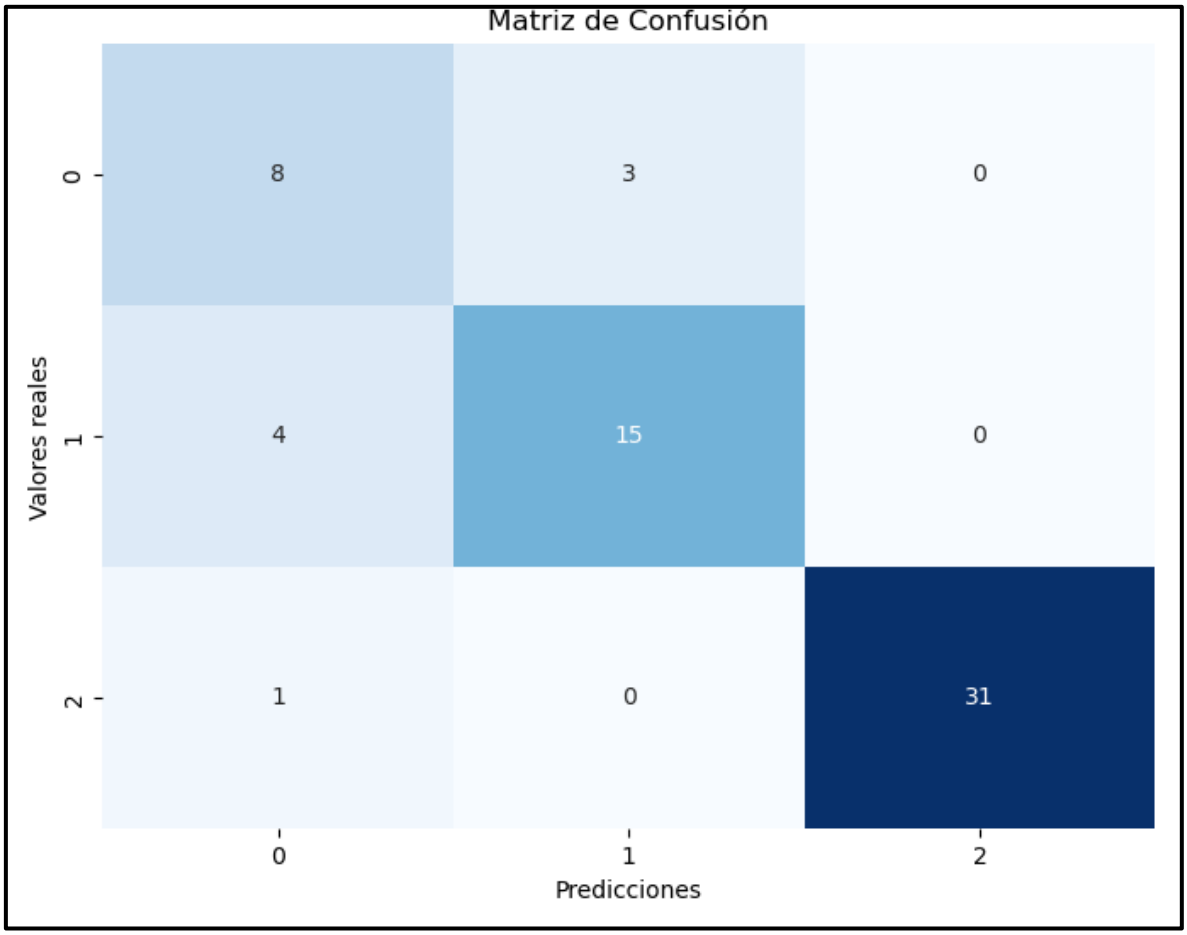
Clases	Medidas			
	TP	TN	FP	FN
0	9	44	7	2
1	13	39	4	6
2	29	30	0	3

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 9 personas que fueron clasificadas con Hernia, 13 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 29 personas fueron clasificadas con Espondilolistesis.

- **Gradient Boosting**

Figura Nº 9: Matriz de Confusión – GB



Fuente: elaboración propia

Tabla Nº 7: Matriz de Observación - GB

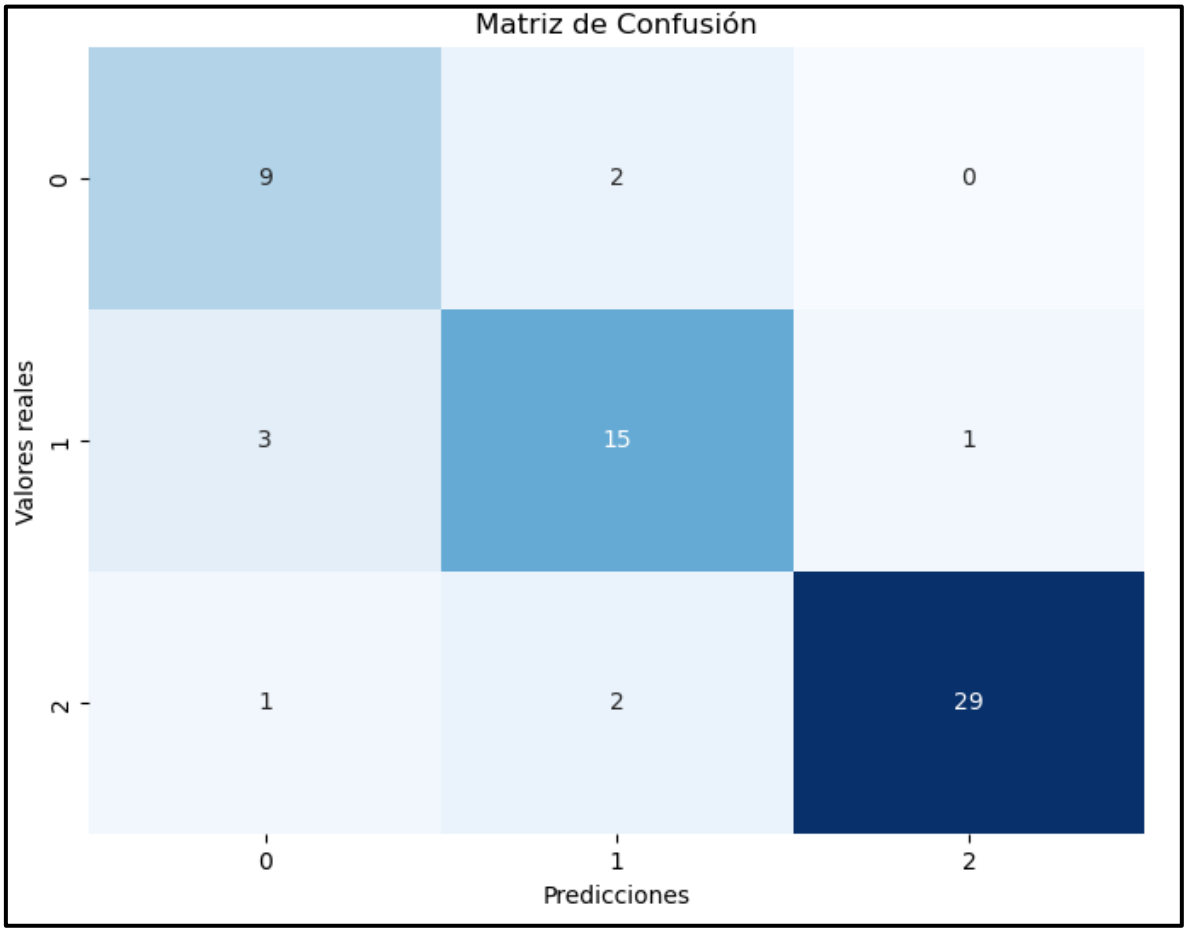
Clases	Medidas			
	TP	TN	FP	FN
0	8	46	5	3
1	15	40	3	4
2	31	30	0	1

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 8 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 31 personas fueron clasificadas con Espondilolistesis.

- **K-Nearest Neighbors**

Figura Nº 10: Matriz de Confusión – KNN



Fuente: elaboración propia

Tabla Nº 8: Matriz de Observación - KNN

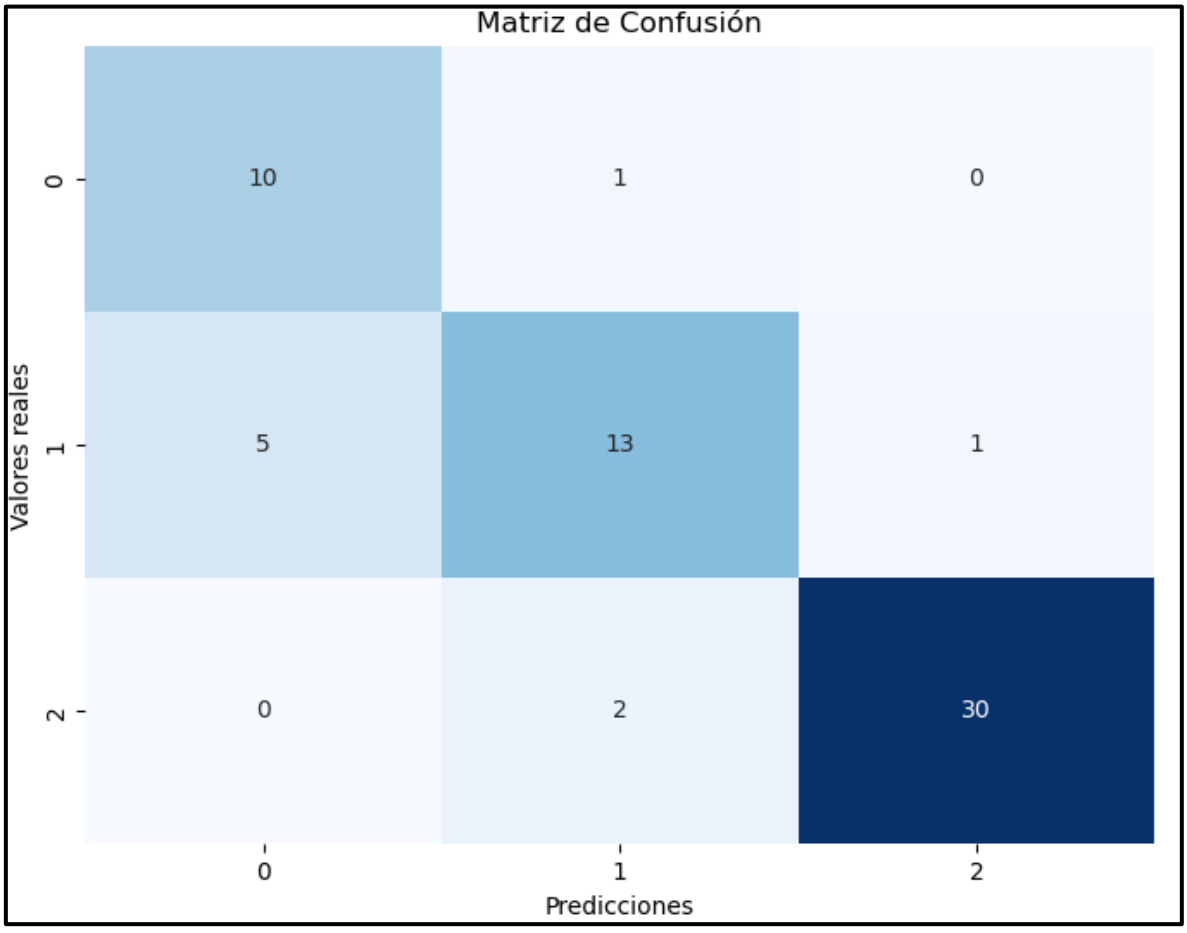
Clases	Medidas			
	TP	TN	FP	FN
0	9	47	4	2
1	15	39	4	4
2	29	29	1	3

Fuente: Elaboración propia

En la aplicación del algoritmo K-Nearest Neighbor, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 29 personas fueron clasificadas con Espondilolistesis.

- **Naive Bayes**

Figura Nº 11: Matriz de Confusión – Naive Bayes



Fuente: elaboración propia

Tabla Nº 9: Matriz de Observación - Naive Bayes

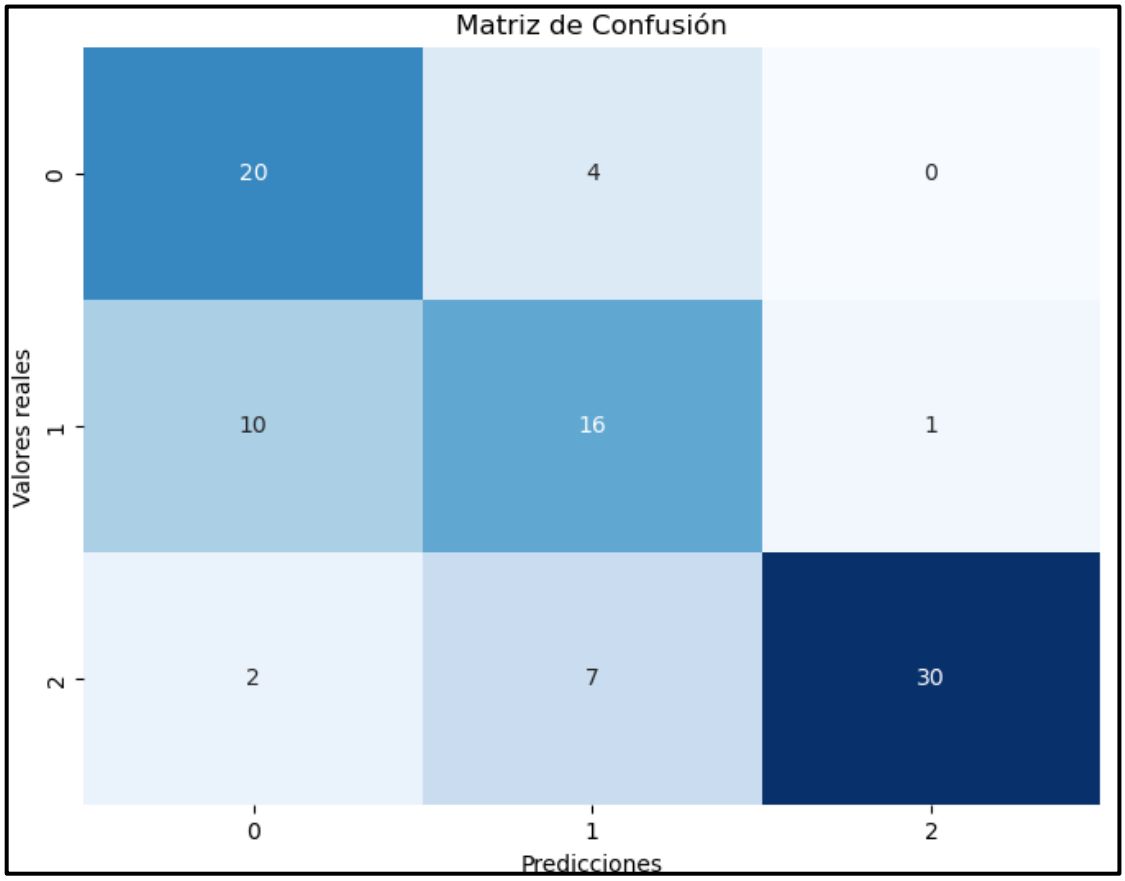
Clases	Medidas			
	TP	TN	FP	FN
0	10	46	5	1
1	13	40	3	6
2	30	29	1	2

Fuente: Elaboración propia

En la aplicación del algoritmo Naive Bayes, se identificó a 10 personas que fueron clasificadas con Hernia, 13 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 30 personas fueron clasificadas con Espondilolistesis.

- **Nearest Centroid**

Figura Nº 12: Matriz de Confusión – Nearest Centroid



Fuente: elaboración propia

Tabla Nº 10: Matriz de Observación - Nearest Centroid

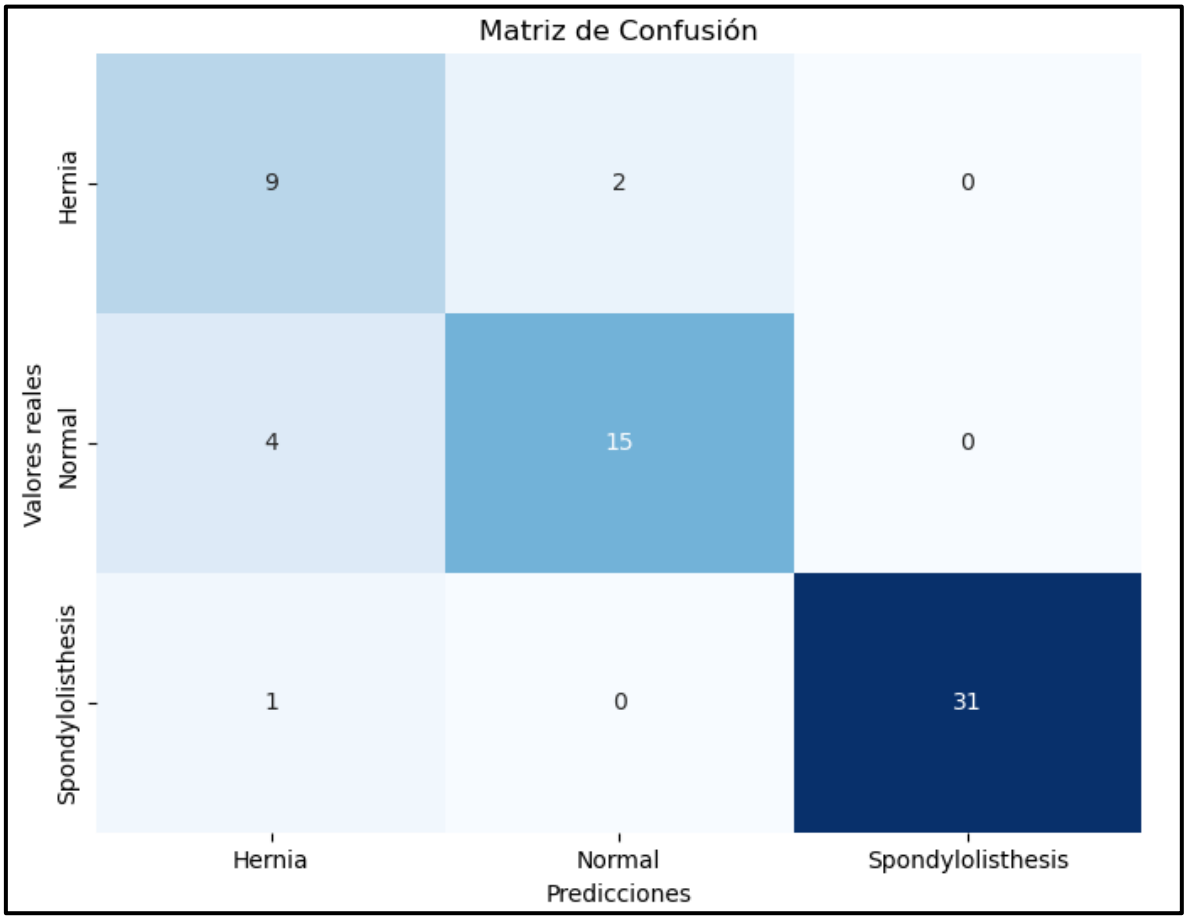
Clases	Medidas			
	TP	TN	FP	FN
0	20	54	12	4
1	16	52	11	11
2	30	50	1	9

Fuente: Elaboración propia

En la aplicación del algoritmo Nearest Centroid, se identificó a 20 personas que fueron clasificadas con Hernia, 16 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 30 personas fueron clasificadas con Espondilolistesis.

- **Random Forest**

Figura Nº 13: Matriz de Confusión – Random Forest



Fuente: elaboración propia

Tabla Nº 11: Matriz de Observación - Random Forest

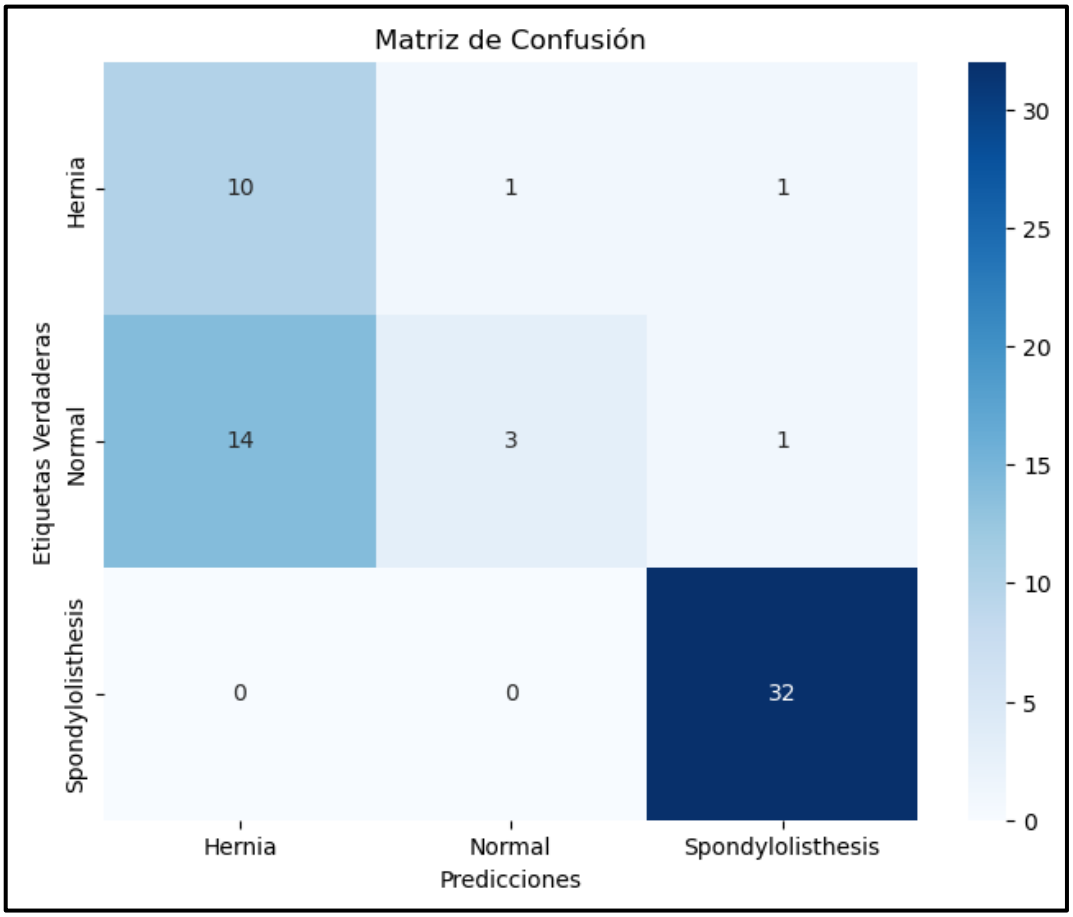
Clases	Medidas			
	TP	TN	FP	FN
0	9	46	5	2
1	15	41	2	4
2	31	30	0	1

Fuente: Elaboración propia

En la aplicación del algoritmo Random Forest, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 31 personas fueron clasificadas con Espondilolistesis.

- **Redes Neuronales**

Figura Nº 14: Matriz de Confusión – Redes Neuronales



Fuente: elaboración propia

Tabla Nº 12: Matriz de Observación - Red Neuronal

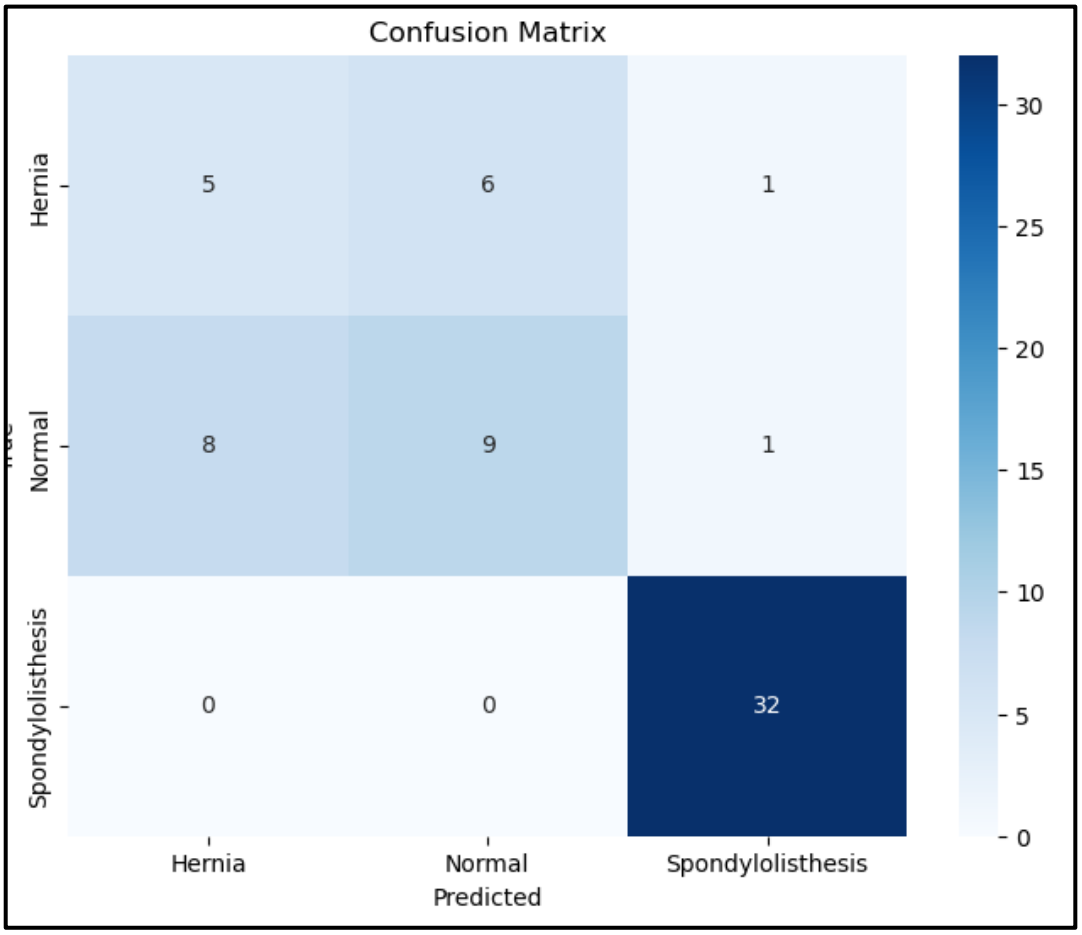
Clases	Medidas			
	TP	TN	FP	FN
0	10	36	14	2
1	3	43	1	15
2	32	28	2	0

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 10 personas que fueron clasificadas con Hernia, 3 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 32 personas fueron clasificadas con Espondilolistesis.

- **Redes neuronales convolucionales**

Figura Nº 15: Matriz de Confusión – Redes neuronales convolucionales



Fuente: elaboración propia

Tabla Nº 13: Matriz de Observación - Redes neuronales convolucionales

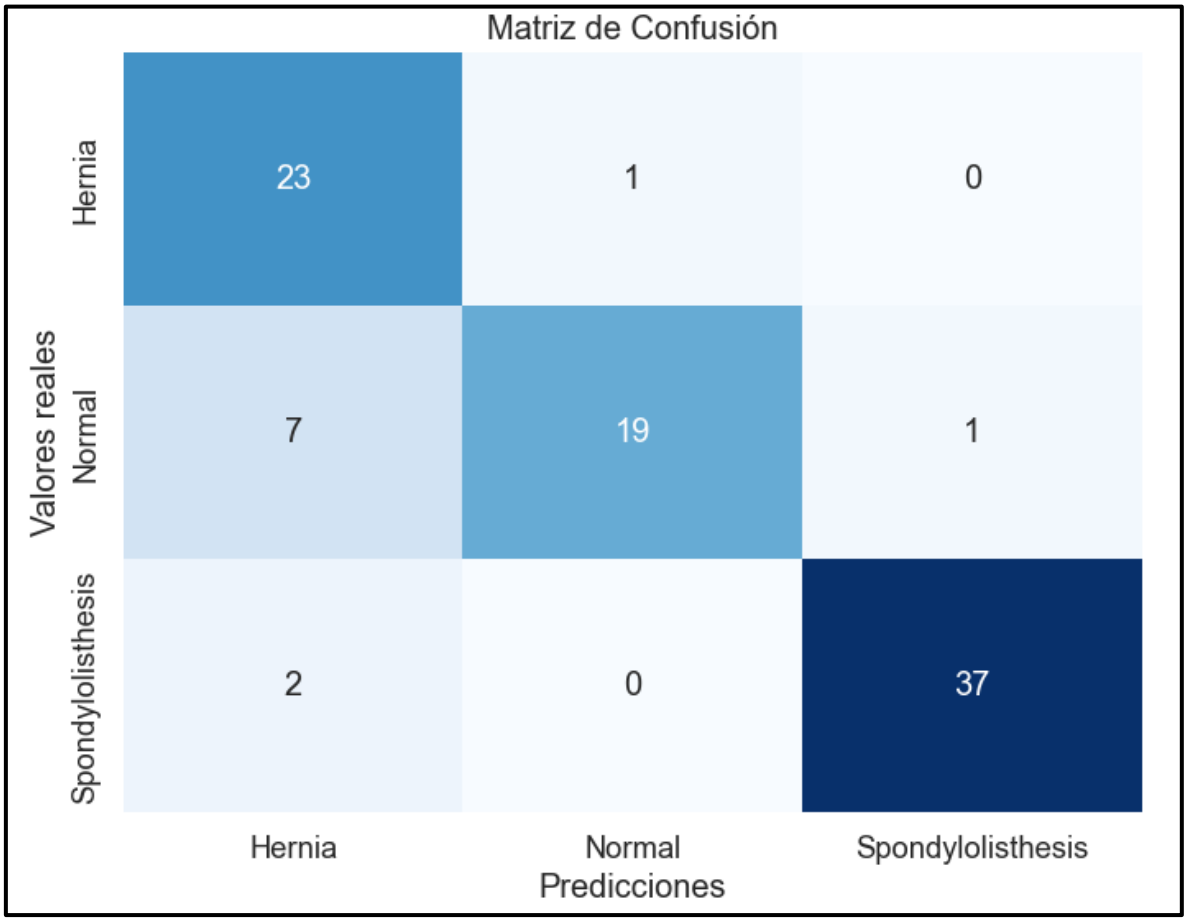
Clases	Medidas			
	TP	TN	FP	FN
0	5	42	8	7
1	9	38	6	9
2	32	28	2	0

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 5 personas que fueron clasificadas con Hernia, 9 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 32 personas fueron clasificadas con Espondilolistesis.

- **Regresión Logística**

Figura Nº 16: Matriz de Confusión – Regresión Logística



Fuente: elaboración propia

Tabla Nº 14: Matriz de Observación - Regresión Logística

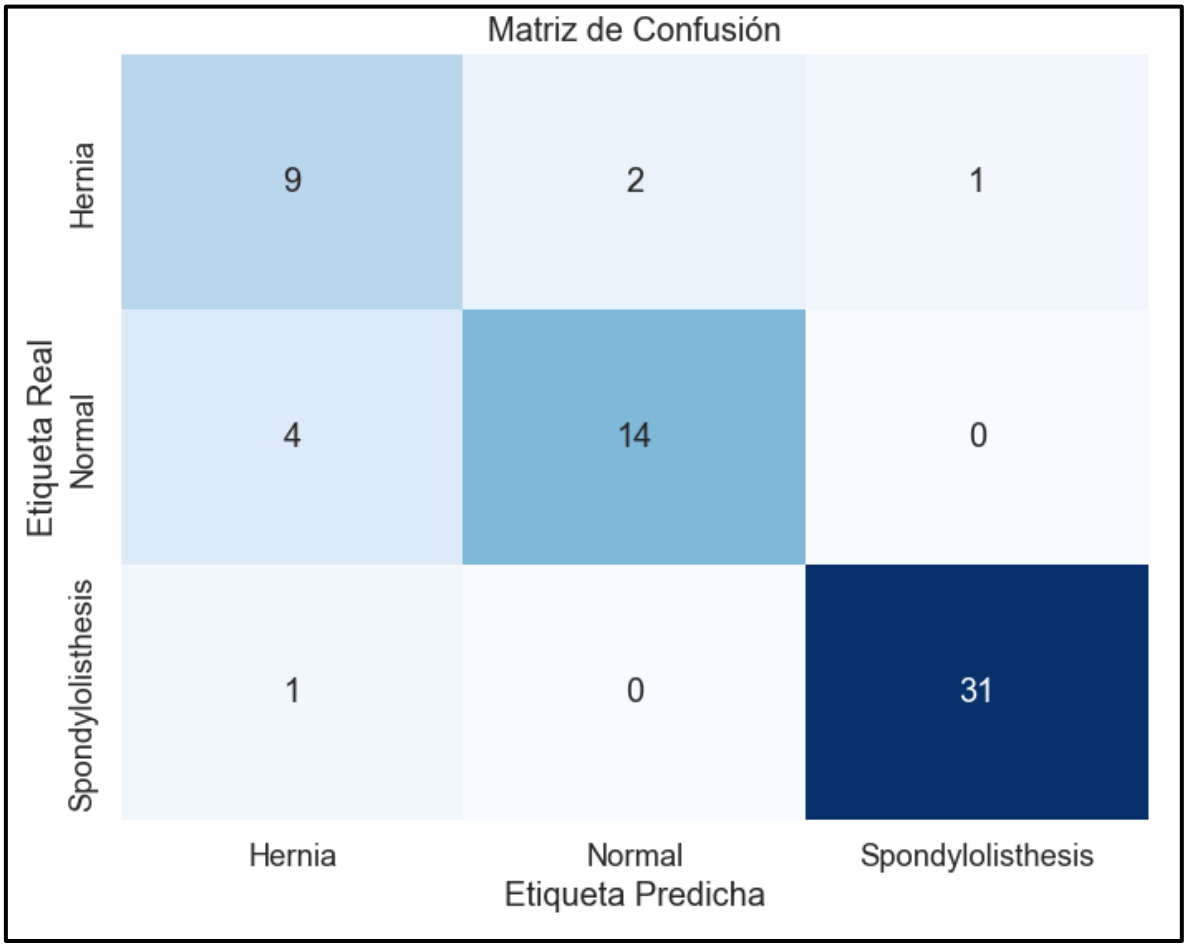
Clases	Medidas			
	TP	TN	FP	FN
0	23	57	9	1
1	19	62	1	8
2	37	50	1	2

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 23 personas que fueron clasificadas con Hernia, 19 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 37 personas fueron clasificadas con Espondilolistesis.

- **Support Vector Machine (SVM)**

Figura Nº 17: Matriz de Confusión – SVM



Fuente: elaboración propia

Tabla Nº 15: Matriz de Observación - SVM

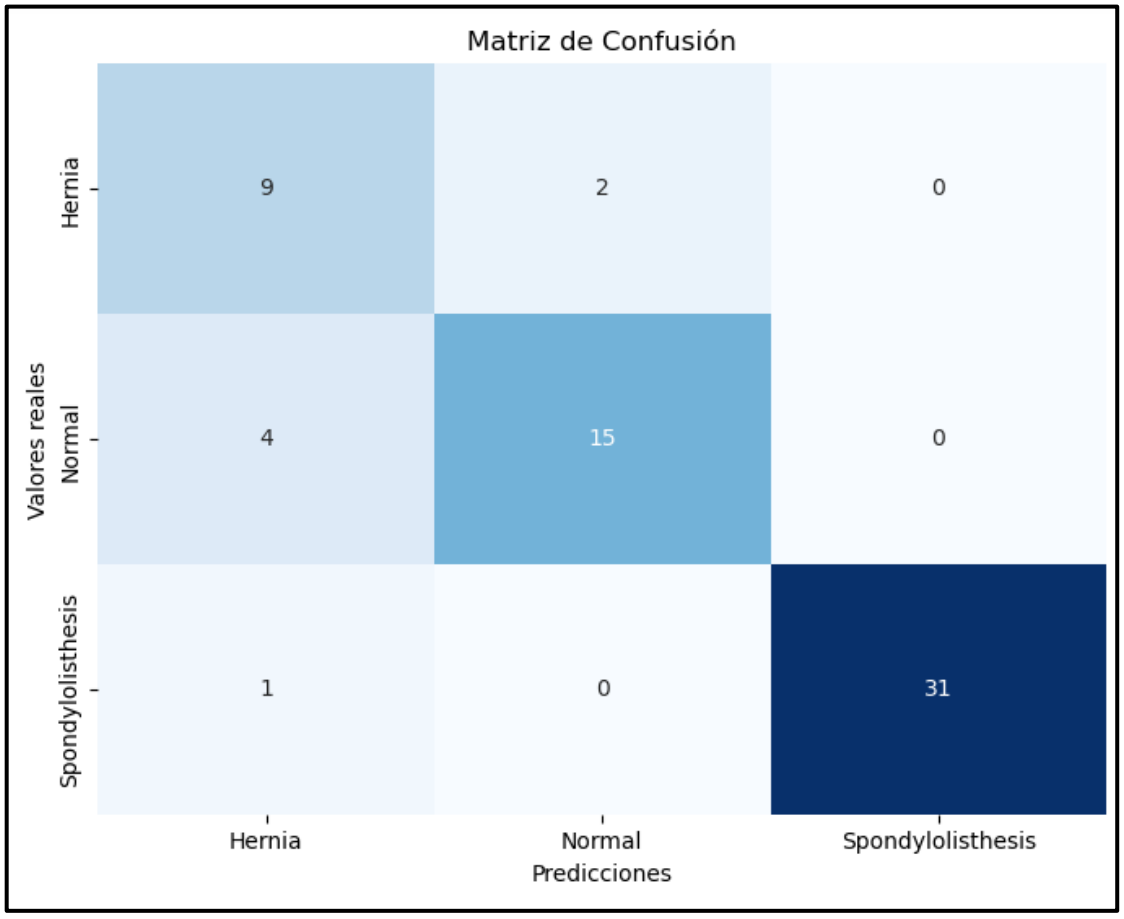
Clases	Medidas			
	TP	TN	FP	FN
0	8	45	4	4
1	14	42	4	4
2	32	29	0	0

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 8 personas que fueron clasificadas con Hernia, 14 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 32 personas fueron clasificadas con Espondilolistesis.

- **XGBoost**

Figura Nº 18: Matriz de Confusión – XGBoost



Fuente: elaboración propia

Tabla Nº 16: Matriz de Observación - XGBoost

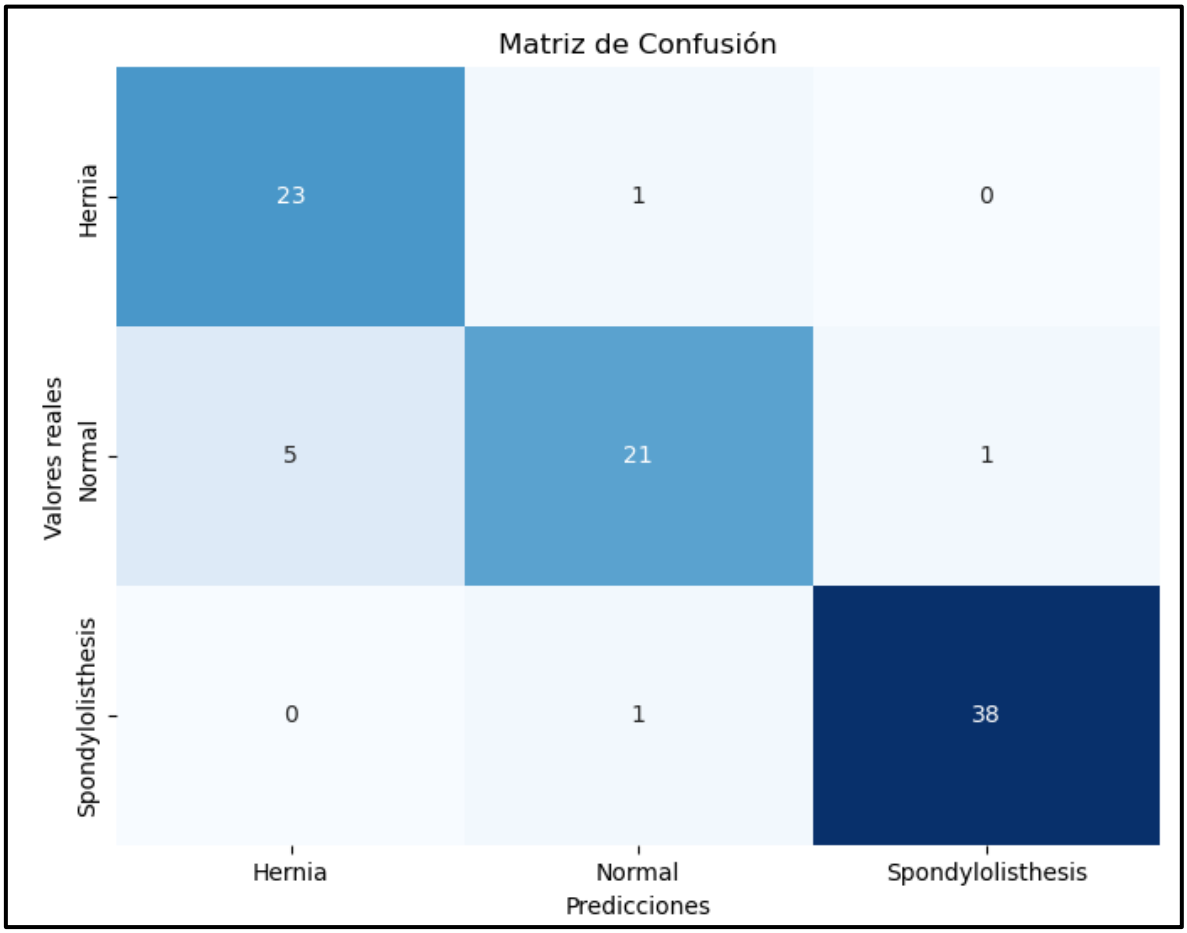
Clases	Medidas			
	TP	TN	FP	FN
0	9	46	5	2
1	15	41	2	4
2	31	30	0	1

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 31 personas fueron clasificadas con Espondilolistesis.

- **AdaBoost**

Figura Nº 19: Matriz de Confusión – AdaBoost



Fuente: elaboración propia

Tabla Nº 17: Matriz de Observación - AdaBoost

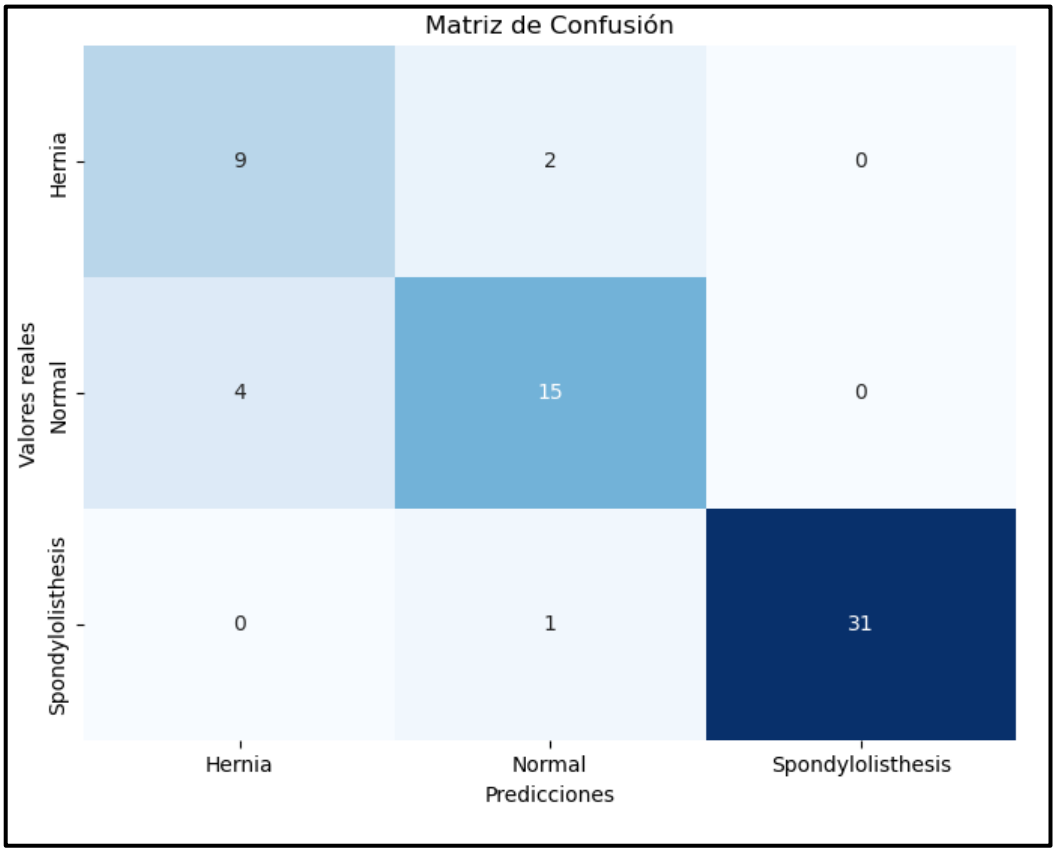
Clases	Medidas			
	TP	TN	FP	FN
0	23	61	5	1
1	21	61	2	6
2	38	50	1	1

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 23 personas que fueron clasificadas con Hernia, 21 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 38 personas fueron clasificadas con Espondilolistesis.

- **Stacking 1**

Figura Nº 20: Matriz de Confusión – Stacking 1



Fuente: elaboración propia

Tabla Nº 18: Matriz de Observación - Stacking 1

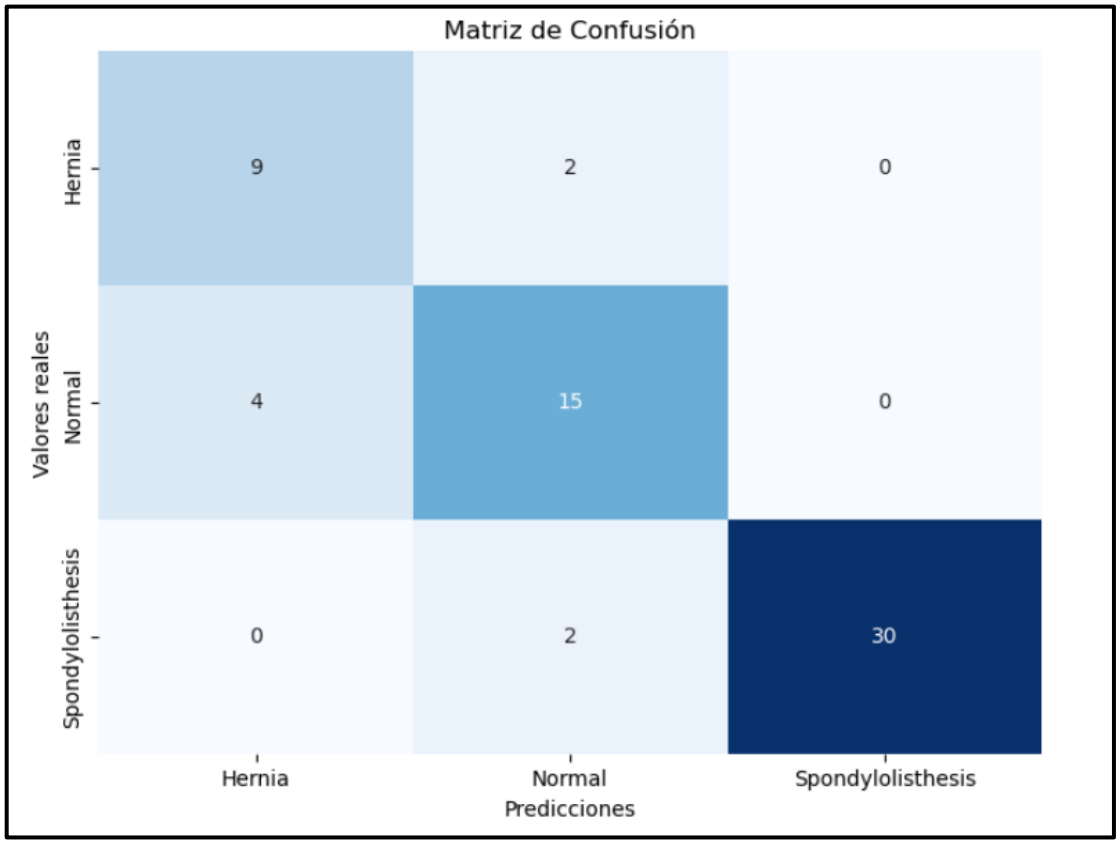
Clases	Medidas			
	TP	TN	FP	FN
0	9	47	4	2
1	15	40	3	4
2	31	30	0	1

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 31 personas fueron clasificadas con Espondilolistesis.

- **Stacking 2**

Figura Nº 21: Matriz de Confusión - Stacking 2



Fuente: elaboración propia

Tabla Nº 19: Matriz de Observación - Stacking 2

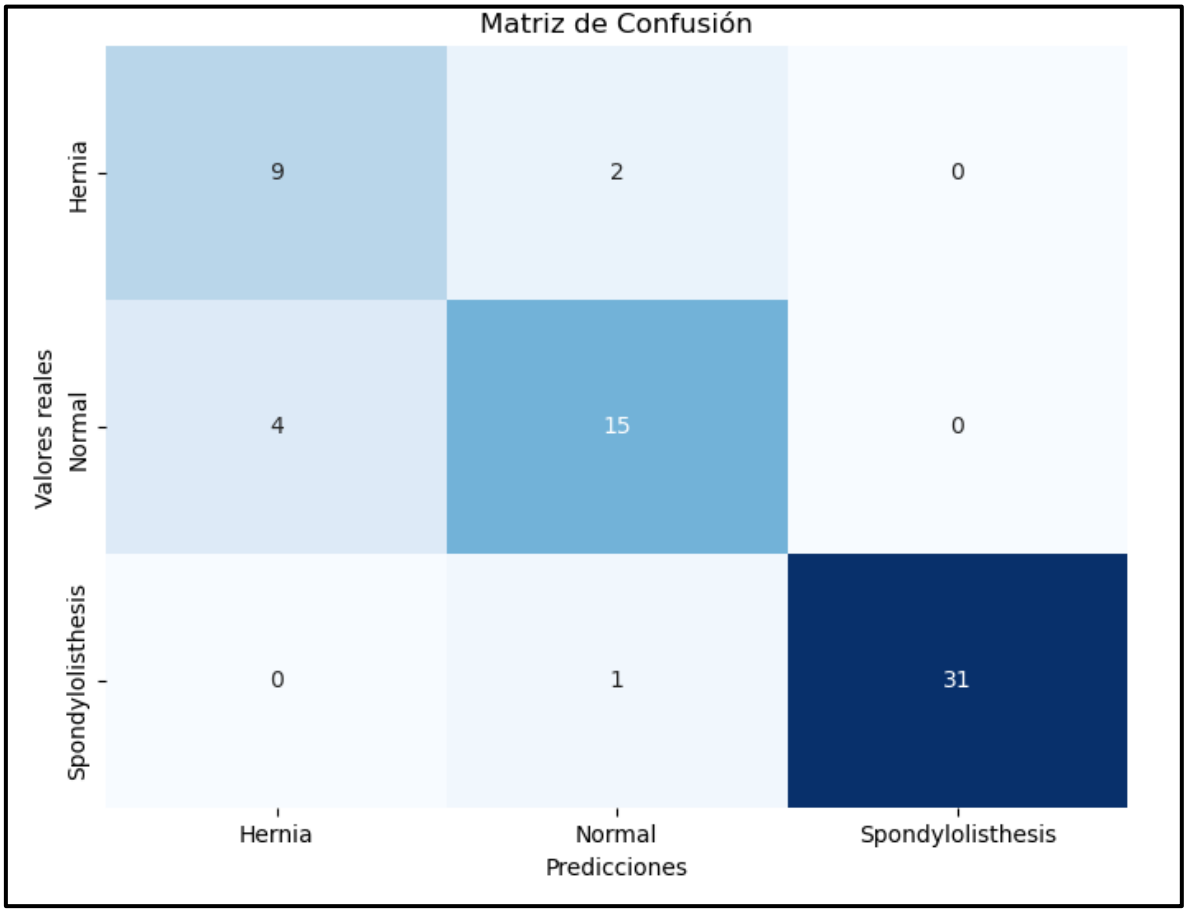
Clases	Medidas			
	TP	TN	FP	FN
0	9	48	4	2
1	15	39	4	4
2	30	30	0	2

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 30 personas fueron clasificadas con Espondilolistesis.

- **Stacking 3**

Figura Nº 22: Matriz de Confusión - Stacking 3



Fuente: elaboración propia

Tabla Nº 20: Matriz de Observación - Stacking 3

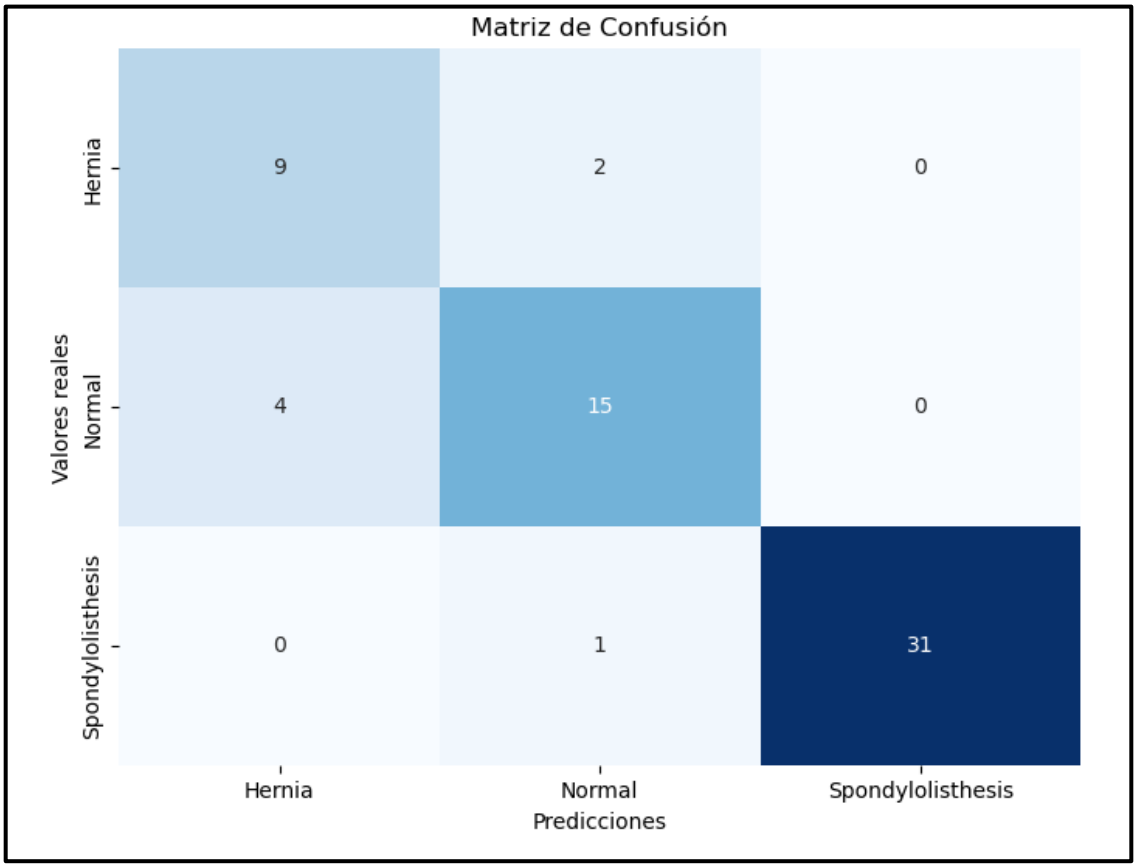
Clases	Medidas			
	TP	TN	FP	FN
0	9	47	4	2
1	15	40	3	4
2	31	30	0	1

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 31 personas fueron clasificadas con Espondilolistesis.

- **Stacking 4**

Figura Nº 23: Matriz de Confusión - Stacking 4



Fuente: elaboración propia

Tabla Nº 21: Matriz de Observación - Stacking 4

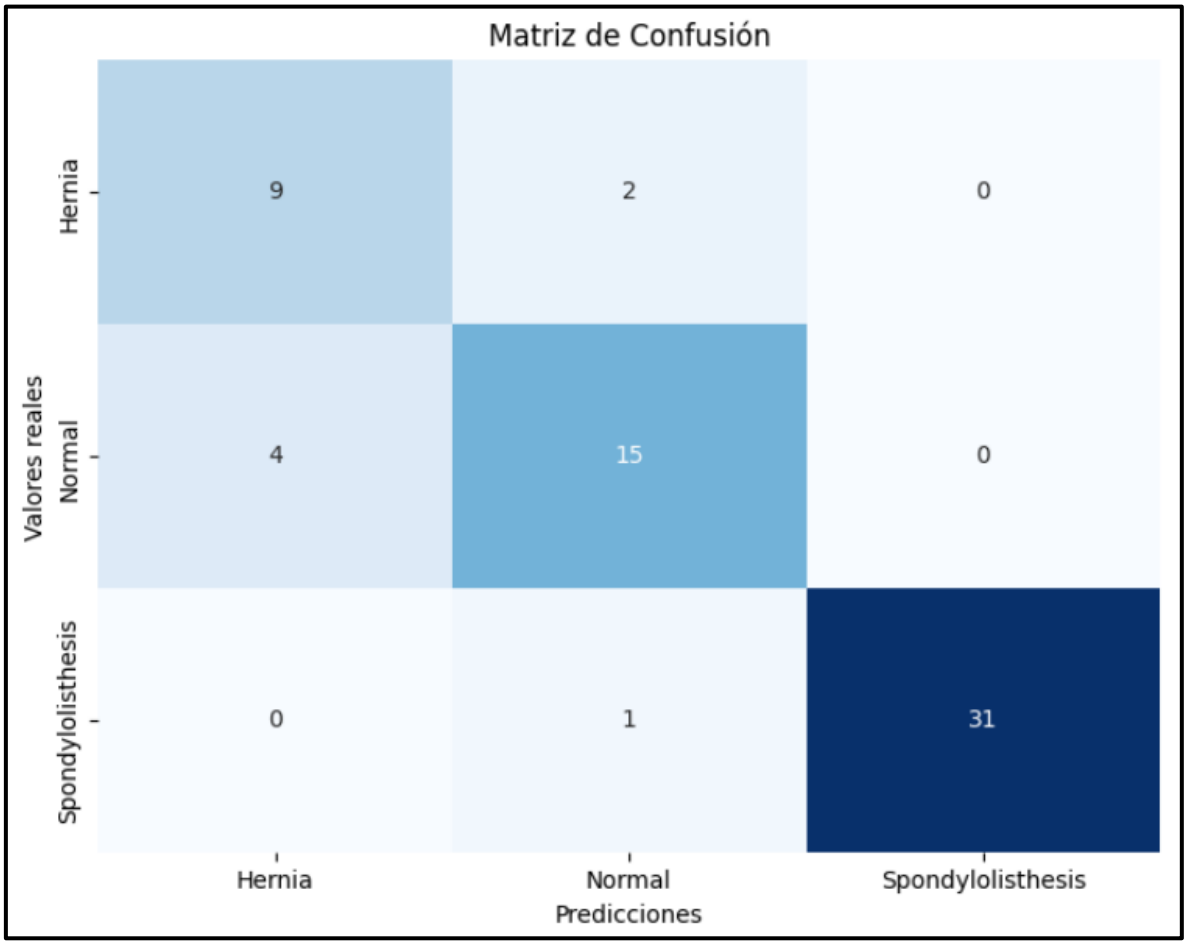
Clases	Medidas			
	TP	TN	FP	FN
0	9	47	4	2
1	15	40	3	4
2	31	30	0	1

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 31 personas fueron clasificadas con Espondilolistesis.

- **Stacking 5**

Figura Nº 24: Matriz de Confusión - Stacking 5



Fuente: elaboración propia

Tabla Nº 22: Matriz de Observación - Stacking 5

Clases	Medidas			
	TP	TN	FP	FN
0	9	47	4	2
1	15	40	3	4
2	31	30	0	1

Fuente: Elaboración propia

En la aplicación del algoritmo Decision Tree, se identificó a 9 personas que fueron clasificadas con Hernia, 15 personas fueron clasificadas con Normal (no padece ninguna enfermedad vertebral) y 31 personas fueron clasificadas con Espondilolistesis.

Objetivo 1: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la exactitud (Accuracy)

- **Decision Tree**

Tabla N° 23: Cálculo de la exactitud con el algoritmo - DT

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	82.26%
	(9+13+29)/(9+13+29+7+4+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 82.26% usando el algoritmo Decision Tree.

- **Gradient Boosting**

Tabla N° 24: Cálculo de la exactitud con el algoritmo - GB

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	87.10%
	(8+15+31)/(8+15+31+5+3+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 87.10% usando el algoritmo Gradient Boosting.

- **KNN**

Tabla N° 25: Cálculo de la exactitud con el algoritmo - KNN

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	85.48%
	(9+15+29)/(9+15+29+4+4+1)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 85.48% usando el algoritmo KNN.

- **Naive Bayes**

Tabla N° 26: Cálculo de la exactitud con el algoritmo - NB

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	85.48%
	(10+13+30)/(10+13+30+5+3+1)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 85.48% usando el algoritmo Naive Bayes.

- **Nearest Centroid**

Tabla N° 27: Cálculo de la exactitud con el algoritmo - NC

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	73.33%
	(20+16+30)/(20+16+30+12+11+1)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una exactitud de 73.33% usando el algoritmo Nearest Centroid

- **Random Forest**

Tabla N° 28: Cálculo de la exactitud con el algoritmo - RF

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	88.71%
	(9+15+31)/(9+15+31+5+2+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 88.71% usando el algoritmo Random Forest.

- **Redes Neuronales**

Tabla N° 29: Cálculo de la exactitud con el algoritmo - RN

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	72.58%
	(10+3+32)/(10+3+32+14+1+2)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una exactitud de 77.42% usando el algoritmo Redes Neuronales.

- **Redes neuronales convolucionales**

Tabla N° 30: Cálculo de la exactitud con el algoritmo - RNC

Clases	Acuraccy **(TP+TN)/(TP+TN+FP+FN) * 100**	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	74.19%
	(5+9+32)/(5+9+32+8+6+2)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una exactitud de 82.26% usando el algoritmo redes neuronales convolucionales

- **Regresión Logística**

Tabla N° 31: Cálculo de la exactitud con el algoritmo - RL

Clases	Acuraccy **(TP+TN)/(TP+TN+FP+FN) * 100**	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	85.48%
	(8+14+31)/(8+14+31+6+3+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una exactitud de 85.48% usando el algoritmo Regresión Logística.

- **Support Vector Machine**

Tabla N° 32: Cálculo de la exactitud con el algoritmo - SVM

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	87.10%
	(9+14+31)/(9+14+31+5+2+1)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 87.10% usando el algoritmo Machine Vector Support.

- **XGBoost**

Tabla N° 33: Cálculo de la exactitud con el algoritmo - XGB

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	88.71%
	(9+15+31)/(9+15+31+5+2+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 88.71% usando el algoritmo XGBoost

- **AdaBoost**

Tabla N° 34: Cálculo de la exactitud con el algoritmo - ADA

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	85.48%
	(8+14+31)/(8+14+31+6+3+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una exactitud de 91.11% usando el algoritmo AdaBoost.

- **Stacking 1**

Tabla N° 35: Cálculo de la exactitud con el modelo Stacking 1

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	88.71%
	(9+15+31)/(9+15+31+4+3+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 88.71% usando el algoritmo Stacking 1

- **Stacking 2**

Tabla N° 36: Cálculo de la exactitud con el modelo Stacking 2

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	87.10%
	(9+16+30)/(9+16+30+3+4+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 88.71% usando el algoritmo Stacking 2.

- **Stacking 3**

Tabla N° 37: Cálculo de la exactitud con el modelo Stacking 3

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	88.71%
	(9+15+31)/(9+15+31+4+3+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 88.71% usando el algoritmo Stacking 3.

- **Stacking 4**

Tabla N° 38: Cálculo de la exactitud con el modelo Stacking 4

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	88.71%
	(9+15+31)/(9+15+31+4+3+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 88.71% usando el Stacking 4.

- **Stacking 5**

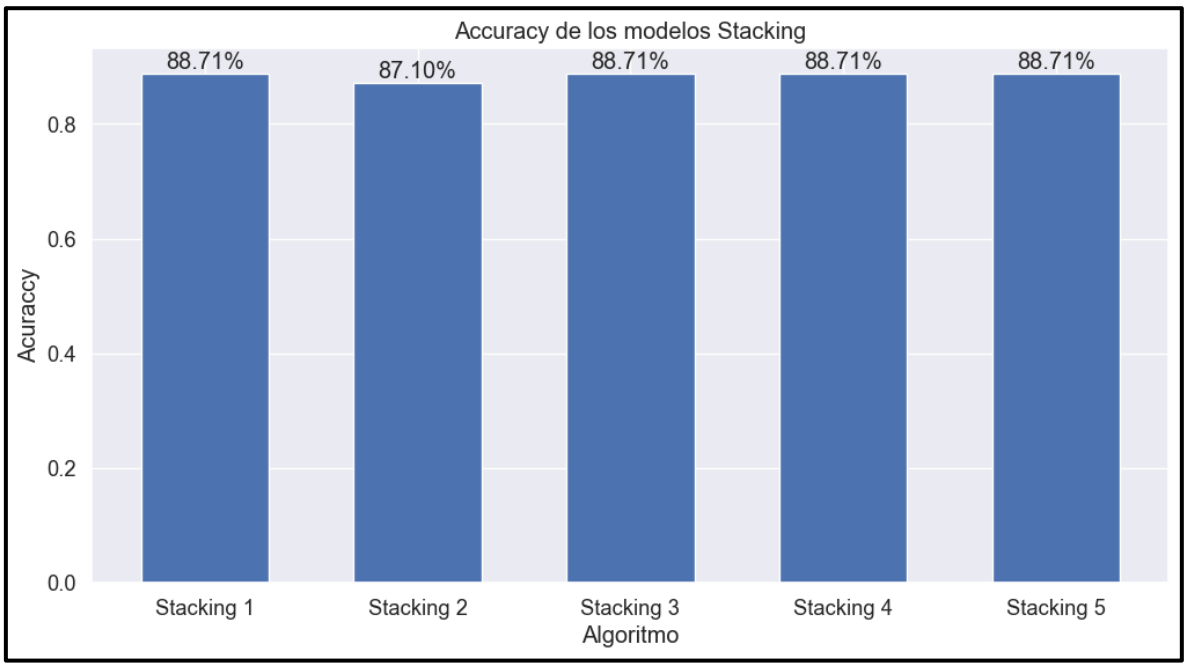
Tabla N° 39: Cálculo de la exactitud con el modelo Stacking 5

Clases	Acuraccy (TP+TN)/(TP+TN+FP+FN) * 100	Resultado
Combinación de clases	(TP_0+TP_1+TP_2)/ (TP_0+TP_1+TP_2+FP_0+FP_1+FP_2) *100	88.71%
	(9+15+31)/(9+15+31+4+3+0)*100	

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 88.71%usando el Stacking 5.

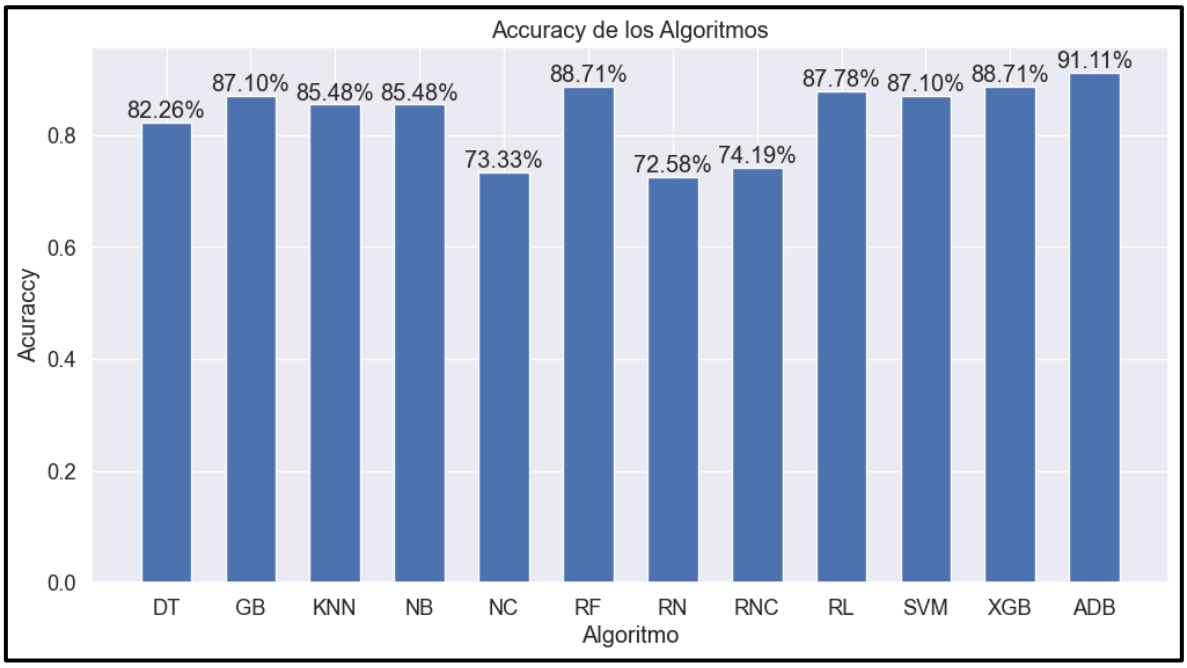
Figura Nº 25: Acuraccy de los modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que los modelos stacking 1, stacking 3, stacking 4 y stacking 5 obtuvieron el mismo porcentaje, con un valor de 88.71%. Por otro lado, el stacking 2 solo obtuvo un 87.10% de exactitud.

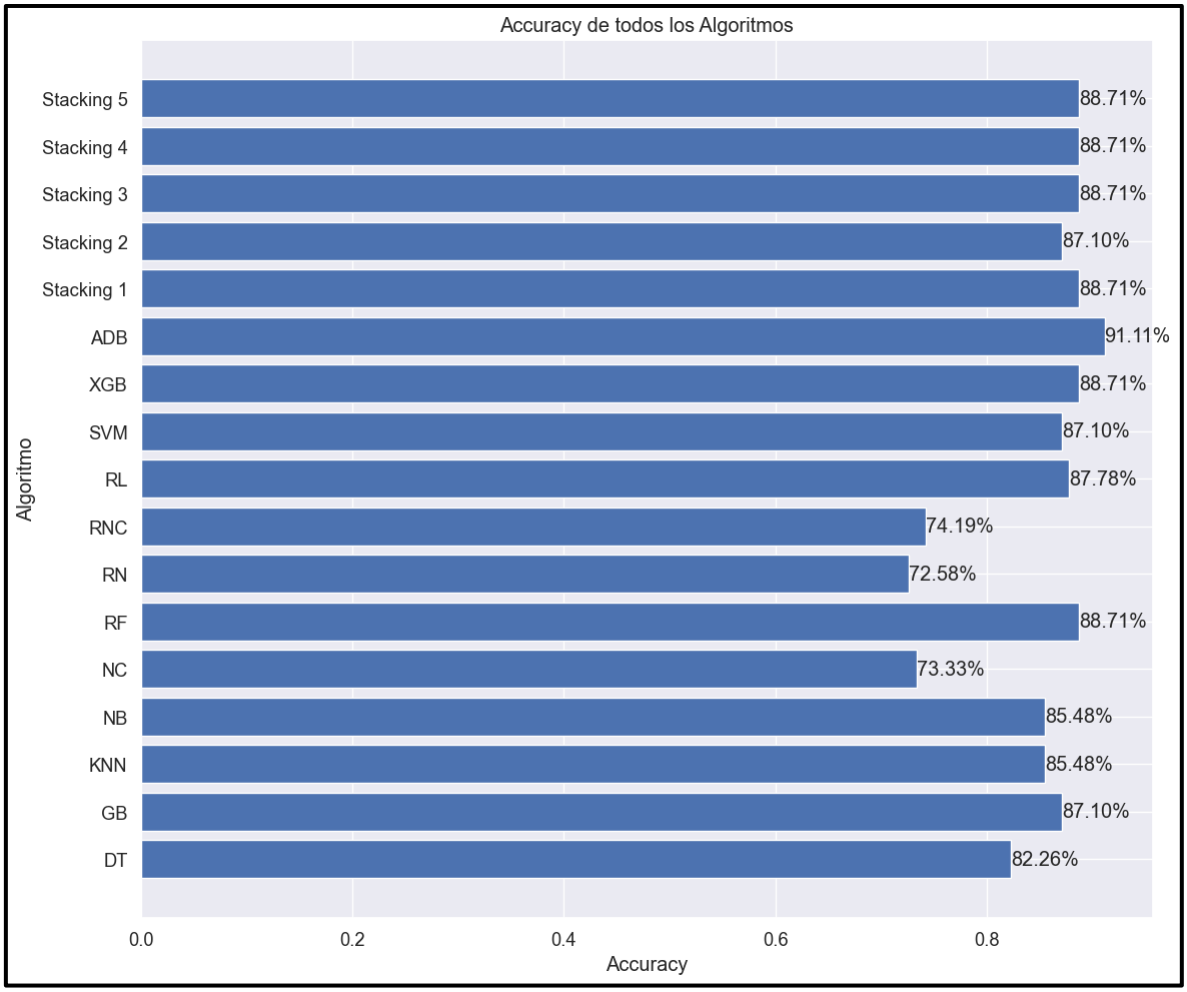
Figura Nº 26: Acuraccy de los algoritmos



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que obtuvieron un mejor Acuraccy son AdaBoost (91.11%), Random Forest (88.71%) y XGBoost (88.71%). Por otro lado, los algoritmos Nearest Centroid (73.33%) y Redes Neuronales (72.58%) obtuvieron el menor porcentaje de exactitud.

Figura Nº 27: Acuraccy de los algoritmos y modelos Stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que obtuvieron un mejor Acuraccy son AdaBoost (91.11%), Random Forest (88.71%) y XGBoost (88.71%). Por otro lado, los stacking 1, stacking 3, stacking 4, stacking 5 tuvieron un 88.71% de exactitud. Realizando una comparación, los modelos stacking obtuvieron la misma exactitud que los modelos XGBoost y Random Forest.

Objetivo 2: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la precisión.

- **Decision Tree**

Tabla Nº 40: Calculo de precisión con el algoritmo - DT

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 7))* 100	56,25%
1	(13/(13 + 4))* 100	76.47%
2	(29/(29 + 0))* 100	100%
Total		77.57%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 77.57% usando el algoritmo Decision Tree.

- **Gradient Boosting**

Tabla Nº 41: Calculo de precisión con el algoritmo - GB

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(8/(8 + 5))* 100	61.54%
1	(15/(15 + 3))* 100	83.33%
2	(31/(31 + 0))* 100	100%
Total		81.62%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 81.62% usando el algoritmo Gradient Boosting.

- **KNN**

Tabla Nº 42: Calculo de precisión con el algoritmo - KNN

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 4))* 100	69.23%
1	(15/(15 + 4))* 100	78.94%
2	(29/(29 + 1))* 100	96.66%
Total		81.61%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 81.61% usando el algoritmo KNN.

- **Naive Bayes**

Tabla Nº 43: Calculo de precisión con el algoritmo - NB

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(10/(10 + 5))* 100	66.67%
1	(13/(13 + 3))* 100	81.25%
2	(30/(30 + 1))* 100	96.77%
Total		81.56%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 81.56% usando el algoritmo Naive Bayes

- **Nearest Centroid**

Tabla Nº 44: Calculo de precisión con el algoritmo - NC

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(20/(20 + 12))* 100	62.5%
1	(16/(16 + 11))* 100	59.26%
2	(30/(30 + 1))* 100	96.77%
Total		72.84%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una precisión de 72.84% usando el algoritmo Nearest Centroid

- **Random Forest**

Tabla Nº 45: Calculo de precisión con el algoritmo - RF

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 5))* 100	64.29%
1	(15/(15 + 2))* 100	88.24%
2	(31/(31 + 0))* 100	100%
Total		84.17%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 84.17% usando el algoritmo Decision Tree.

- **Redes Neuronales**

Tabla Nº 46: Calculo de precisión con el algoritmo - RN

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(10/(10 + 14))* 100	41.67%
1	(3/(3 + 1))* 100	75%
2	(32/(32 + 2))* 100	94.12%
Total		70.26%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 70.26% usando el algoritmo Redes Neuronales.

- **Redes neuronales convolucionales**

Tabla Nº 47: Calculo de precisión con el algoritmo - RNC

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(5/(5 + 8))* 100	38.46%
1	(9/(9 + 6))* 100	60%
2	(32/(32 + 2))* 100	94.12%
Total		64.19%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 64.19% usando el algoritmo redes neuronales convolucionales.

- **Regresión Logística**

Tabla Nº 48: Calculo de precisión con el algoritmo - RL

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(23/(23 + 9))* 100	71.88%
1	(19/(19 + 1))* 100	95%
2	(37/(37 + 1))* 100	97.37%
Total		88.08%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 88.08% usando el algoritmo de Regresión Logística

- **Support Vector Machine (SVM)**

Tabla Nº 49: Calculo de precisión con el algoritmo - SVM

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 5))* 100	64.29%
1	(14/(14 + 2))* 100	87.5%
2	(31/(31 + 1))* 100	96.88%
Total		82.89%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 82.89% usando el algoritmo SVM.

- **XGBoost**

Tabla Nº 50: Calculo de precisión con el algoritmo - XGB

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 5))* 100	64.29%
1	(15/(15 + 2))* 100	88.24%
2	(31/(31 + 0))* 100	100%
Total		84.17%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 84.17% usando el algoritmo Decision Tree.

- **AdaBoost**

Tabla Nº 51: Calculo de precisión con el algoritmo - ADA

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(23/(23 + 5))* 100	82.14%
1	(21/(21 + 2))* 100	91.3%
2	(38/(38 + 1))* 100	97.44%
Total		90.29%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 90.29% usando el algoritmo Decision Tree.

- **Stacking 1**

Tabla Nº 52: Calculo de precisión con el modelo Stacking 1

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 4))* 100	69.23%
1	(15/(15 + 3))* 100	83.33%
2	(31/(31 + 0))* 100	100%
Total		84.19%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 84,19% usando el modelo Stacking 1.

- **Stacking 2**

Tabla Nº 53: Calculo de precisión con el modelo Stacking 2

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 4))* 100	69.23%
1	(15/(15 + 4))* 100	78.95%
2	(30/(30 + 0))* 100	100%
Total		82.73%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 82.73% usando el modelo Stacking 2.

- **Stacking 3**

Tabla Nº 54: Calculo de precisión con el modelo Stacking 3

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 4))* 100	69.23%
1	(15/(15 + 3))* 100	83.33%
2	(31/(31 + 0))* 100	100%
Total		84.19%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 84.19% usando el modelo Stacking 3.

- **Stacking 4**

Tabla Nº 55: Calculo de precisión con el modelo Stacking 4

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 4))* 100	69.23%
1	(15/(15 + 3))* 100	83.33%
2	(31/(31 + 0))* 100	100%
Total		84.19%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 89.43% usando el a modelo Stacking 4.

- **Stacking 5**

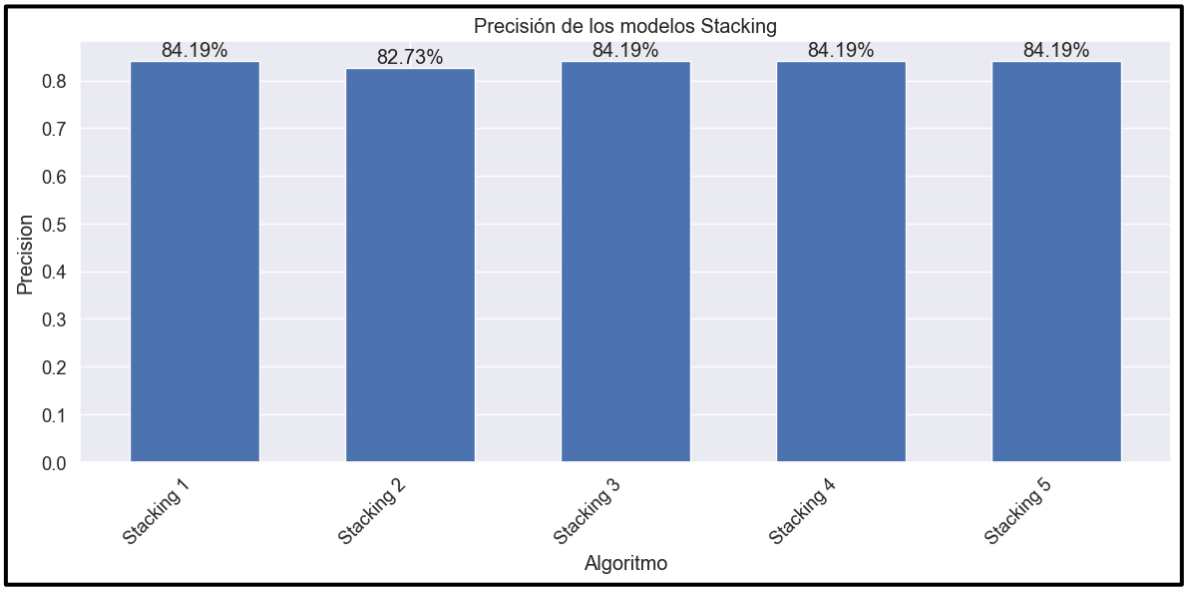
Tabla Nº 56: Calculo de precisión con el modelo Stacking 5

Clases	Precisión (TP/(TP + FP))* 100	Resultado
0	(9/(9 + 4))* 100	69.23%
1	(15/(15 + 3))* 100	83.33%
2	(31/(31 + 0))* 100	100%
Total		84.19%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una precisión de 89.43% usando el modelo Stacking 5.

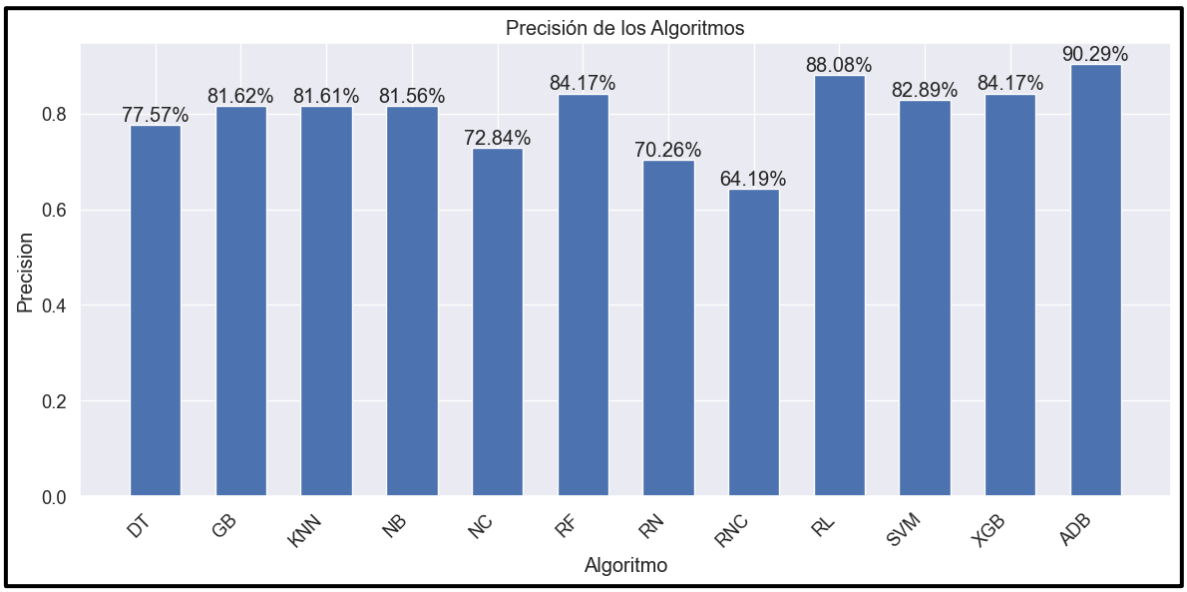
Figura Nº 28: Precisión de los modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que los modelos stacking 1, stacking 3, stacking 4 y stacking 5 obtuvieron el mismo porcentaje, con un valor de 84.19%. Por otro lado, el stacking 2 solo obtuvo un 82.73% de precision.

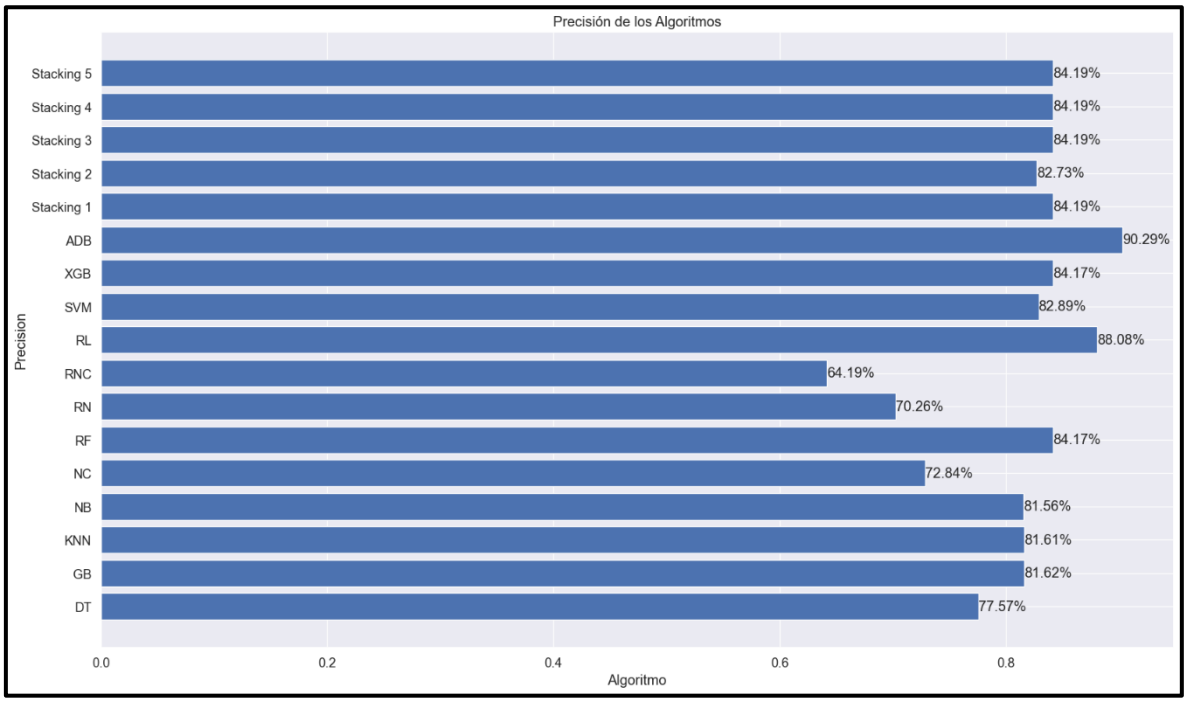
Figura Nº 29: Precisión de los algoritmos



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que obtuvieron una mejor precisión son AdaBoost (90.29%), Random Forest (85.07%), Naive Bayes (83.34%) y XGBoost (84.17%)

Figura Nº 30: Precisión de los algoritmos y modelos stacking



Fuente: Elaboración propia

Interpretación: Los algoritmos con mejor precisión son AdaBoost (90.29%), RL (88.08%) y los modelos stacking 1,3,4 y 5 obtuvieron una precisión de 84.19%.

Objetivo 3: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la recall.

- **Decision Tree**

Tabla Nº 57: Calculo de la Sensibilidad (recall) con el algoritmo - DT

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.81%
1	(13/(13 + 6))* 100	68.42%
2	(29/(29 + 3))* 100	90.62%
Total		80.29%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una recall de 80.29% usando el algoritmo Decision Tree.

- **Gradient Boosting**

Tabla Nº 58: Calculo de la Sensibilidad (recall) con el algoritmo - GB

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(8/(8 + 3))* 100	72.73%
1	(15/(15 + 4))* 100	78.95%
2	(31/(31 + 1))* 100	96.88%
Total		82.85%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una recall de 82.85 % usando el algoritmo Gradient Boosting.

- **KNN**

Tabla Nº 59: Calculo de la Sensibilidad (recall) con el algoritmo - KNN

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.81%
1	(15/(15 + 4))* 100	78.94%
2	(29/(29 + 3))* 100	90.62%
Total		83.80%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 83.80% usando el algoritmo KNN.

- **Naive Bayes**

Tabla Nº 60: Calculo de la Sensibilidad (recall) con el algoritmo - NB

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(10/(10 + 1))* 100	90.61%
1	(13/(13 + 6))* 100	68.42%
2	(30/(30 + 2))* 100	93.75%
Total		84.36%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 84.36% usando el algoritmo Naive Bayes.

- **Nearest Centroid**

Tabla Nº 61: Calculo de la Sensibilidad (recall) con el algoritmo - NC

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(20/(20 + 4))* 100	83.33%
1	(16/(16 + 11))* 100	59.26%
2	(30/(30 + 9))* 100	76.92%
Total		73.17%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con un recall de 73.17% usando el algoritmo Nearest Centroid.

- **Random Forest**

Tabla Nº 62: Calculo de la Sensibilidad (recall) con el algoritmo - RF

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.81%
1	(15/(15 + 4))* 100	78.95%
2	(31/(31 + 1))* 100	96.88%
Total		85.88%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 85.88% usando el algoritmo Random Forest.

- **Redes Neuronales**

Tabla Nº 63: Calculo de la Sensibilidad (recall) con el algoritmo - RN

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(10/(10 + 12)* 100	45.45%
1	(3/(3 + 15))* 100	16.67%
2	(32/(32 + 0))* 100	100%
Total		66.67%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 62.96% usando el algoritmo Redes neuronales

- **Redes neuronales convolucionales**

Tabla Nº 64: Calculo de la Sensibilidad (recall) con el algoritmo - RNC

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(5/(5 + 7))* 100	41.67%
1	(9/(9 + 9))* 100	50%
2	(32/(32 + 0))* 100	100%
Total		63.89%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 63.77% usando el algoritmo redes neuronales convolucionales

- **Regresión Logística**

Tabla Nº 65: Calculo de la Sensibilidad (recall) con el algoritmo - RL

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(23/(23 + 1))* 100	95.83%
1	(19/(19 + 8))* 100	70.37%
2	(37/(37 + 2))* 100	94.87%
Total		87.03%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 87.03% usando el algoritmo Regresión Logística.

- **Support Vector Machine**

Tabla Nº 66: Calculo de la Sensibilidad (recall) con el algoritmo - SVM

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 3))* 100	75%
1	(14/(14 + 4))* 100	77.78%
2	(31/(31 + 1))* 100	96.88%
Total		83.22%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 83.22% usando el algoritmo Support Vector Machine

- **XGBoost**

Tabla Nº 67: Calculo de la Sensibilidad (recall) con el algoritmo - XGB

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.81%
1	(15/(15 + 4))* 100	78.94%
2	(31/(31 + 1))* 100	96.88%
Total		85.88%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 85.88% usando el algoritmo XGBoost.

- **AdaBoost**

Tabla Nº 68: Calculo de la Sensibilidad (recall) con el algoritmo - ADA

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(23/(23 + 1))* 100	95.83%
1	(21/(21 + 6))* 100	77.78%
2	(38/(38 + 1))* 100	97.44%
Total		90.35%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con un recall de 90.35% usando el algoritmo AdaBoost.

- **Stacking 1**

Tabla Nº 69: Calculo de la Sensibilidad (recall) con el modelo Stacking 1

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.82%
1	(15/(15 + 4))* 100	78.95%
2	(31/(31 + 1))* 100	96.87%
Total		85.88%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 85.88% usando el Stacking 1.

- **Stacking 2**

Tabla Nº 70: Calculo de la Sensibilidad (recall) con el modelo Stacking 2

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.82%
1	(16/(16 + 3))* 100	84.21%
2	(30/(30 + 2))* 100	93.75%
Total		86.6%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 86.6% usando el Stacking 2.

- **Stacking 3**

Tabla Nº 71: Calculo de la Sensibilidad (recall) con el modelo Stacking 3

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.82%
1	(15/(15 + 4))* 100	78.95%
2	(31/(31 + 1))* 100	96.87%
Total		85.88%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una recall de 85.88% usando el Stacking 3.

- **Stacking 4**

Tabla Nº 72: Calculo de la Sensibilidad (recall) con el modelo Stacking 4

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.82%
1	(15/(15 + 4))* 100	78.95%
2	(31/(31 + 1))* 100	96.87%
Total		88.71%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 88.71% usando el stacking 4.

- **Stacking 5**

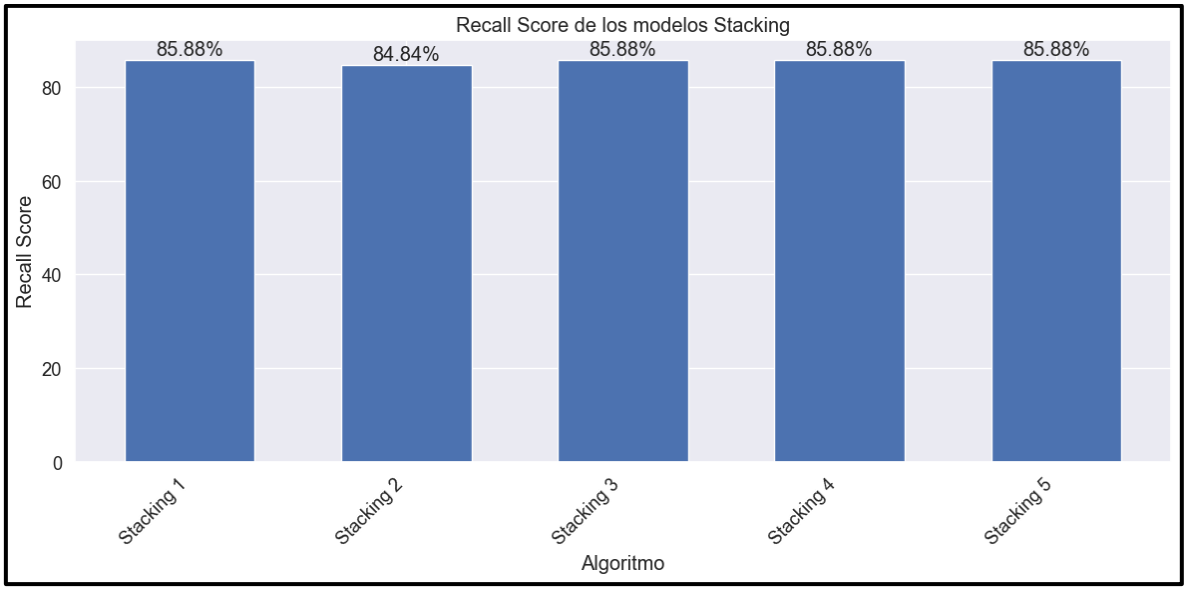
Tabla Nº 73: Calculo de la Sensibilidad (recall) con el modelo Stacking 5

Clases	Recall (TP/(TP + FN))* 100	Resultado
0	(9/(9 + 2))* 100	81.82%
1	(15/(15 + 4))* 100	78.95%
2	(31/(31 + 1))* 100	96.87%
Total		88.71%

Fuente: Elaboración propia

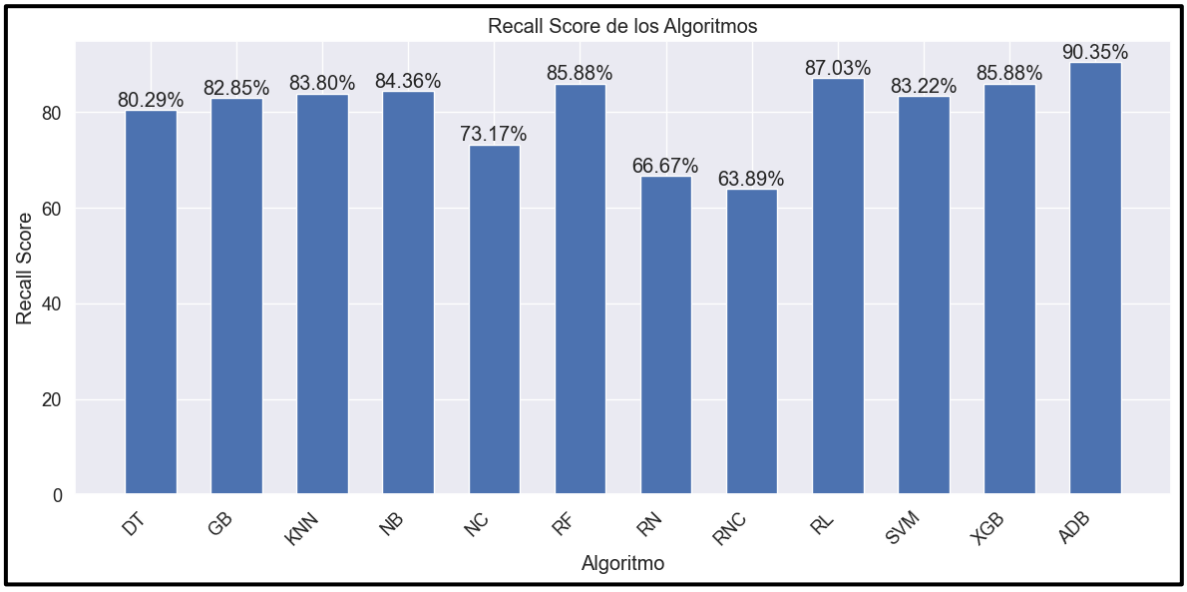
Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un recall de 88.71% usando el stacking 5.

Figura Nº 31: Recall de los Algoritmos



Interpretación: Se visualiza que los modelos stacking 1, stacking 3, stacking 4 y stacking 5 obtuvieron el mismo porcentaje, con un valor de 85.88%. Por otro lado, el stacking 2 solo obtuvo un 84.84% de precisión.

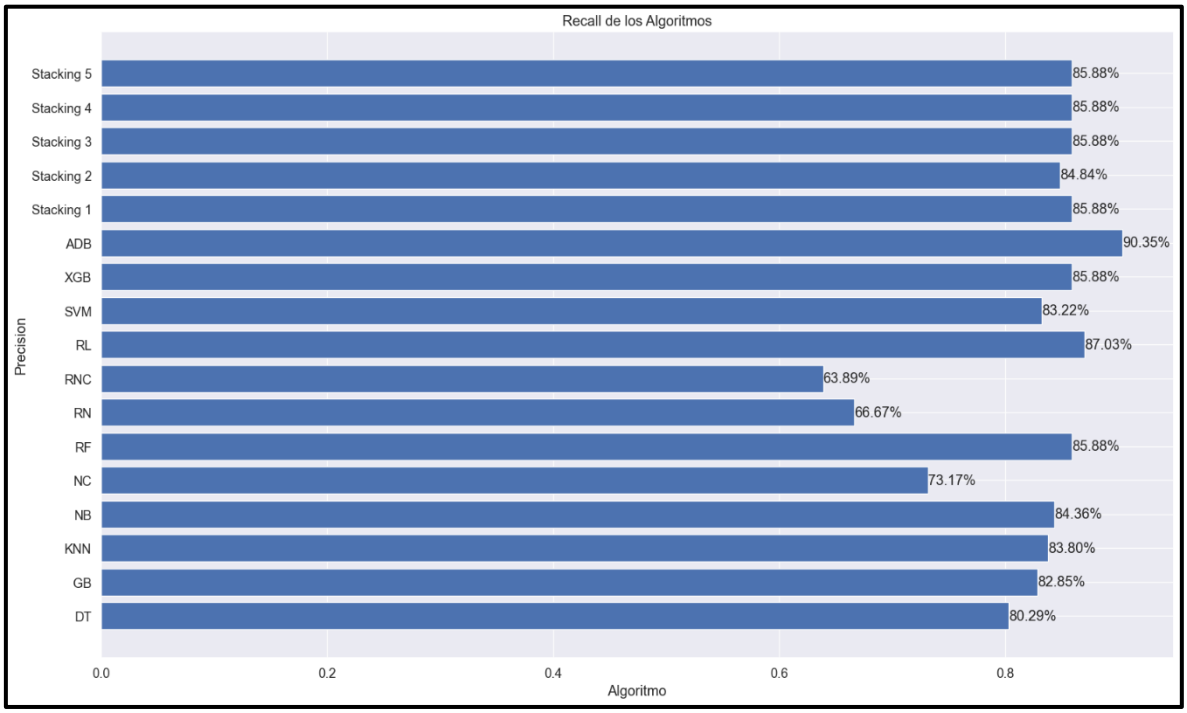
Figura Nº 32: Recall de los Algoritmos



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que tuvieron un mejor porcentaje recall son AdaBoost (90.35%), Regresión Logística (87.03%) y los modelos Random Forest y XGBoost obtuvieron el mismo porcentaje de 85.88% de recall.

Figura Nº 33: Recall de los Algoritmos y modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que tuvieron un mejor porcentaje recall son AdaBoost (90.35%), Regresión Logística (87.03%). Por otro lado, los modelos RF, XGB y stacking 1, stacking 3, stacking 4 y stacking 5obtuvieron el mismo porcentaje de 85.88% de recall.

Objetivo 4: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la F1-score

- **Decision Tree**

Tabla Nº 74: Calculo de F1-score con el algoritmo - DT

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.5625* 0.8181)/ 0.5625 + 0.8181)*100	66.78%
1	(2*(0.7647 * 0.6842)/ 0.7647+ 0.6842)*100	72.20%
2	(2*(1 * 0.9062)/ 1+ 0.9062)*100	95.10%
Total		77,9%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 77.9% usando el algoritmo Decision Tree.

- **Gradient Boosting**

Tabla Nº 75: Calculo de F1-score con el algoritmo - GB

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6154* 0.7273/ 0.6154 + 0.7273)*100	66.61%
1	(2*(0.8333 * 0.7895)/ 0.8333 + 0.7895)*100	81.01%
2	(2*(1 * 0.9688)/ 1 + 0.9688)*100	98.5%
Total		82,05%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 82.05% usando el algoritmo Gradient Boosting.

- **KNN**

Tabla Nº 76: Calculo de F1-score con el algoritmo - KNN

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6923* 0.8181/ 0.6923 + 0.8181)*100	74.98%
1	(2*(0.7894 * 0.7894)/ 0.7894+ 0.7894)*100	78.96%
2	(2*(0.9666 * 0.9062)/ 0.9666 + 0.9062)*100	93.59%
Total		92,50%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 92.50% usando el algoritmo KNN.

- **Naive Bayes**

Tabla Nº 77: Calculo de F1-score con el algoritmo - NB

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6667* 0.9061/ 0.6667 + 0.9061)*100	76.88%
1	(2*(0.8125 * 0.6842)/ 0.8125 + 0.6842)*100	74.19%
2	(2*(0.9677 * 0.9375)/ 0.9677 + 0.9375)*100	95.21%
Total		82.15%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 82.15% usando el algoritmo Naive Bayes.

- **Nearest Centroid**

Tabla Nº 78: Calculo de F1-score con el algoritmo - NC

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.625 * 0.8333/ 0.625 + 0.8333)*100	71.44%
1	(2*(0.5926 * 0.5926)/ 0.5926 + 0.5926)*100	59.18%
2	(2*(0.9677 * 0.7692)/ 0.9677 + 0.7692)*100	85.86%
Total		72.13%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 72.13% usando el algoritmo Nearest Centroid

- **Random Forest**

Tabla Nº 79: Calculo de F1-score con el algoritmo - RF

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6429* 0.8181/ 0.6429 + 0.8181)*100	71.92%
1	(2*(0.8824 * 0.7895)/ 0.8824 + 0.7895)*100	83.17%
2	(2*(1 * 0.9688)/ 1 + 0.9688)*100	98.55%
Total		84.58%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 84.58% usando el algoritmo Random Forest

- **Red Neuronal**

Tabla Nº 80: Calculo de F1-score con el algoritmo -RN

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.4167 * 0.4545/ 0.4167 + 0.4545)*100	43.43%
1	(2*(0.75 * 0.1667)/ 0.75 + 0.1667)*100	27.28%
2	(2*(0.9412 * 1)/ 0.9412 + 1)*100	97.01%
Total		59.93%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 59.93% usando el algoritmo Redes Neuronales

- **Redes neuronales convolucionales**

Tabla Nº 81: Calculo de F1-score con el algoritmo - RNC

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.3846 * 0.4167 / 0.3846 + 0.4167)*100	39.97%
1	(2*(0.6 * 0.5)/ 0.6 + 0.5)*100	54.54%
2	(2*(0.9412 * 1)/ 0.9412 + 1)*100	97.01%
Total		63.84%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 63.84% usando el algoritmo redes neuronales convolucionales.

- **Regresión Logística**

Tabla Nº 82: Calculo de F1-score con el algoritmo - RL

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.7188 * 0.9583/ 0.7188 + 0.9583)*100	82.18%
1	(2*(0.95 * 0.7037)/ 0.95 + 0.7037)*100	81.04%
2	(2*(0.9737 * 0.9487)/ 0.9737 + 0.9487)*100	95.98%
Total		86.37%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 86.37% usando el algoritmo Regresión Logística.

- **Support Vector Machine**

Tabla Nº 83: Calculo de F1-score con el algoritmo - SVM

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6429 * 0.75/ 0.6429 + 0.75)*100	69.24%
1	(2*(0.875 * 0.7778)/ 0.875 + 0.7778)*100	82.34%
2	(2*(0.9688 * 0.9688)/ 0.9688 + 0.9688)*100	96.80%
Total		82.82%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 82.82% usando el algoritmo Support Vector Machine

- **XGBoost**

Tabla Nº 84: Calculo de F1-score con el algoritmo -XGB

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6429 * 0.8181/ 0.6429 + 0.8181)*100	71.92%
1	(2*(0.8824 * 0.7894)/ 0.8824 + 0.7894)*100	83.17%
2	(2*(1 * 0.9687)/ 1+ 0.9687)*100	98.54%
Total		84.58%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 84.58% usando el algoritmo XGBoost.

- **AdaBoost**

Tabla Nº 85: Calculo de F1-score con el algoritmo - ADA

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.8214 * 0.9583/ 0.8214 + 0.9583)*100	88.14%
1	(2*(0.913 * 0.7778)/ 0.913 + 0.7778)*100	84.06%
2	(2*(0.9744 * 0.9744)/ 0.9744 + 0.9744)*100	97.49%
Total		89.97%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 89.97% usando el algoritmo AdaBoost

- **Stacking 1**

Tabla Nº 86: Calculo de F1-score con el modelo Stacking 1

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6923 * 0.8182/ 0.6923 + 0.8182)*100	75.11%
1	(2*(0.8333 * 0.7895)/ 0.8333+ 0.7895)*100	81.01%
2	(2*(1 * 0.9687)/ 1 + 0.9687)*100	98.55%
Total		84.83%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral una puntuación F1-score de 84.43% usando el algoritmo Stacking 1.

- **Stacking 2**

Tabla Nº 87: Calculo de F1-score con el modelo Stacking 2

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6923 * 0.8182/ 0.6923 + 0.8182)*100	75.10%
1	(2*(0.7895 * 0.8421)/ 0.7895 + 0.8421)*100	81.46%
2	(2*(1 * 0.9375)/ 1 + 0.9375)*100	96.92%
Total		83.57%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 83.57% usando el algoritmo Stacking 2.

- **Stacking 3**

Tabla Nº 88: Calculo de F1-score con el modelo Stacking 3

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6923 * 0.8182/ 0.6923 + 0.8182)*100	75.11%
1	(2*(0.8333 * 0.7895)/ 0.8333+ 0.7895)*100	81.01%
2	(2*(1 * 0.9687)/ 1 + 0.9687)*100	98.55%
Total		84.83%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 84.83% usando el algoritmo Stacking 3

- **Stacking 4**

Tabla Nº 89: Calculo de F1-score con el modelo Stacking 4

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6923 * 0.8182/ 0.6923 + 0.8182)*100	75.11%
1	(2*(0.8333 * 0.7895)/ 0.8333+ 0.7895)*100	81.01%
2	(2*(1 * 0.9687)/ 1 + 0.9687)*100	98.55%
Total		84.83%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 84.83% usando el algoritmo Stacking 4

- **Stacking 5**

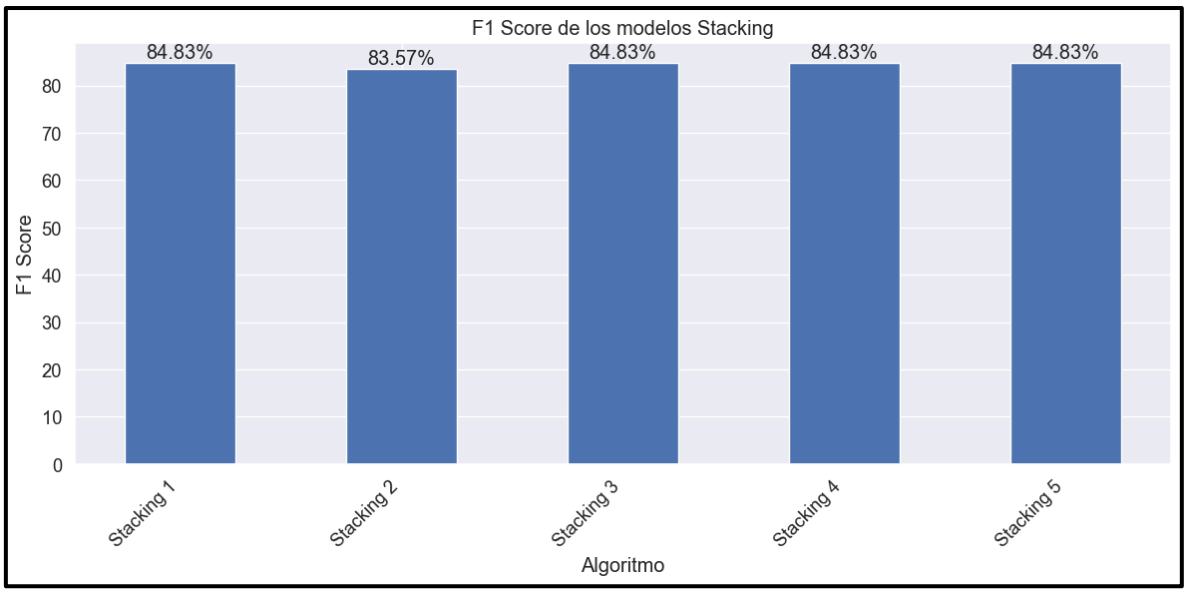
Tabla Nº 90: Calculo de F1-score con el modelo Stacking 5

Clases	F1-score (2*(Precisiòn*Recall)/ Precisión+Recall) * 100	Resultado
0	(2*(0.6923 * 0.8182/ 0.6923 + 0.8182)*100	75.11%
1	(2*(0.8333 * 0.7895)/ 0.8333+ 0.7895)*100	81.01%
2	(2*(1 * 0.9687)/ 1 + 0.9687)*100	98.55%
Total		84.83%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una puntuación F1-score de 84.83% usando el algoritmo Stacking 5.

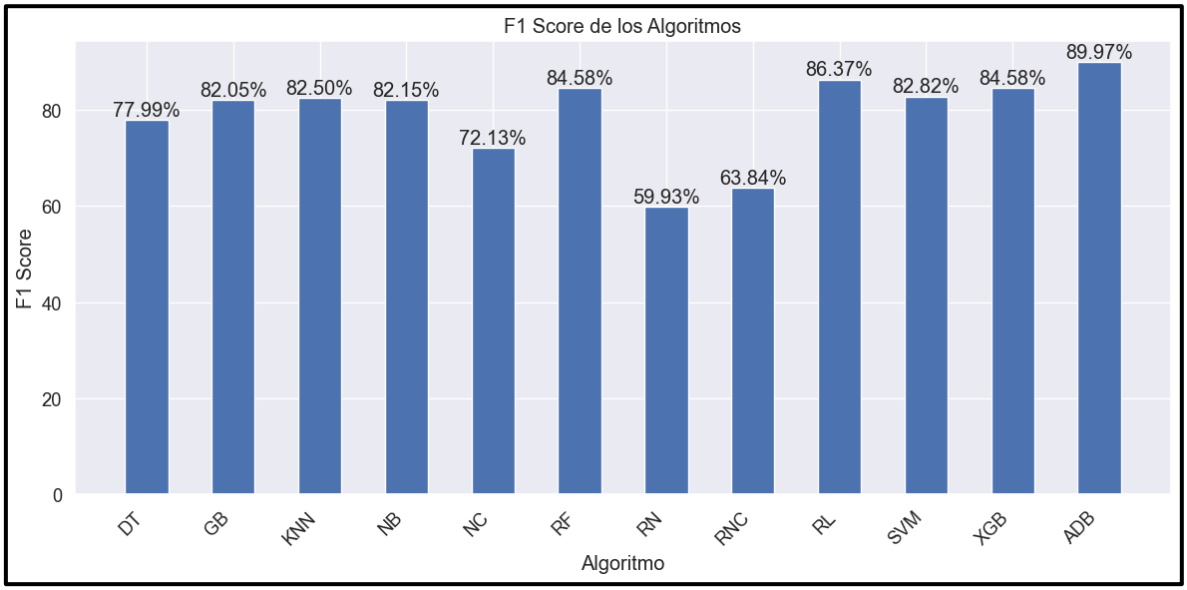
Figura Nº 34: F1-score de los modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que los modelos stacking 1, stacking 3, stacking 4 y stacking 5 obtuvieron el mismo porcentaje, con un valor de 84.83%. Por otro lado, el stacking 2 solo obtuvo un 83.57% de f1-score.

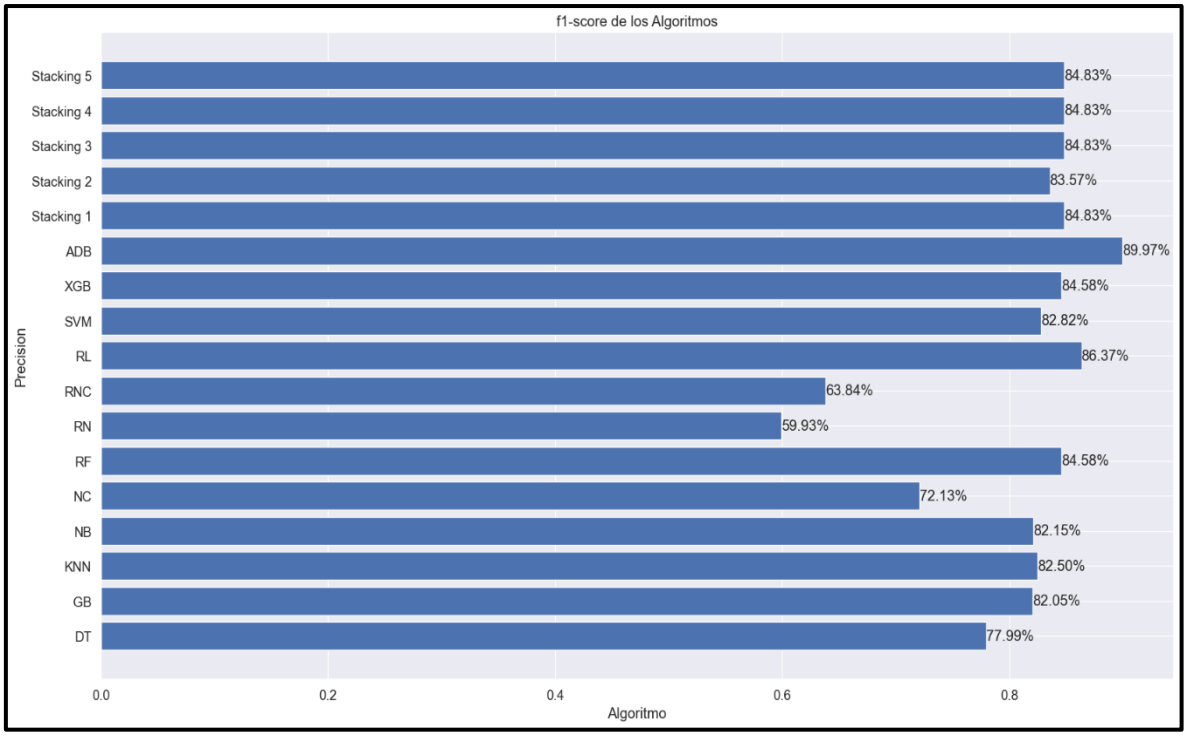
Figura Nº 35: F1-score de los algoritmos



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que tuvieron mejores resultados son AdaBoost (89.97%), Regresión Logística (86.37%) y Random Forest (84.58%).

Figura Nº 36: F1-score de los algoritmos y modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que tuvieron mejores resultados son AdaBoost (89.97%) y Regresión Logística (86.37%).

Objetivo 5: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la AUC.

- **Decision Tree**

Tabla Nº 91: Calculo de AUC con el algoritmo - DT

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i+\text{Sensibilidad}_{i+1})\cdot(1-\text{Especificidad}_{i+1}-1-\text{Especificidad}_i)$	Resultado
Total		0.84

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 81.62% usando el algoritmo Decision Tree.

- **Gradient Boosting**

Tabla Nº 92: Calculo de AUC con el algoritmo - GB

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i+\text{Sensibilidad}_{i+1})\cdot(1-\text{Especificidad}_{i+1}-1-\text{Especificidad}_i)$	Resultado
Total		0.95

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 93.88% usando el algoritmo Gradient Boosting

- **KNN**

Tabla Nº 93: Calculo de AUC con el algoritmo - KNN

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.94

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 95.32% usando el algoritmo KNN.

- **Naive Bayes**

Tabla Nº 94: Calculo de AUC con el algoritmo - NB

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.95

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 94.96% usando el algoritmo Naive Bayes.

- **Random Forest**

Tabla Nº 95: Calculo de AUC con el algoritmo - RF

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.96

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 95.44%% usando el algoritmo Random Forest

- **Redes Neuronales**

Tabla Nº 96: Calculo de AUC con el algoritmo - RN

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.90

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 92.64%% usando el algoritmo Redes Neuronales.

- **Redes neuronales convolucionales**

Tabla Nº 97: Calculo de AUC con el algoritmo - RNC

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.91

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 96.55% usando el algoritmo redes neuronales convolucionales.

- **Regresión Logística**

Tabla Nº 98: Calculo de AUC con el algoritmo - RL

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.96

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo Regresión Logística

- **Support Vector Machine**

Tabla Nº 99: Calculo de AUC con el algoritmo - SVM

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1}) \cdot (1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.96

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 95.94% usando el algoritmo Support Vector Machine

- **XGBoost**

Tabla Nº 100: Calculo de AUC con el algoritmo - XGB

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1}) \cdot (1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.96

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo XGBoost.

- **AdaBoost**

Tabla Nº 101: Calculo de AUC con el algoritmo - ADA

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.98

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo AdaBoost.

- **Stacking 1**

Tabla Nº 102: Calculo de AUC con el modelo Stacking 1

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.4481

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo Stacking 1

- **Stacking 2**

Tabla Nº 103: Calculo de AUC con el modelo Stacking 2

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.4821

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo Stacking 2

- **Stacking 3**

Tabla Nº 104: Calculo de AUC con el modelo Stacking 3

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.4450

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo Stacking 3

- **Stacking 4**

Tabla Nº 105: Calculo de AUC con el modelo Stacking 4

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.50037

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo Stacking 4

- **Stacking 5**

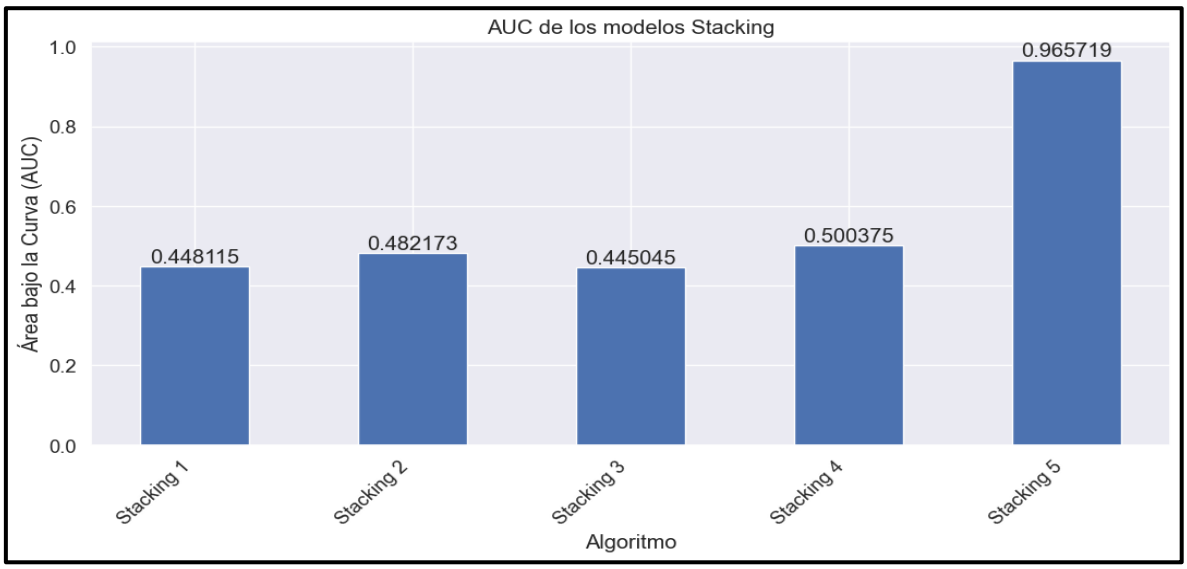
Tabla Nº 106: Calculo de AUC con el modelo Stacking 5

Clases	AUC $\sum_{i=1}^{n-1} 21(\text{Sensibilidad}_i + \text{Sensibilidad}_{i+1})\cdot(1 - \text{Especificidad}_{i+1} - 1 - \text{Especificidad}_i)$	Resultado
Total		0.96571

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un AUC de 0.82% usando el algoritmo Stacking 5.

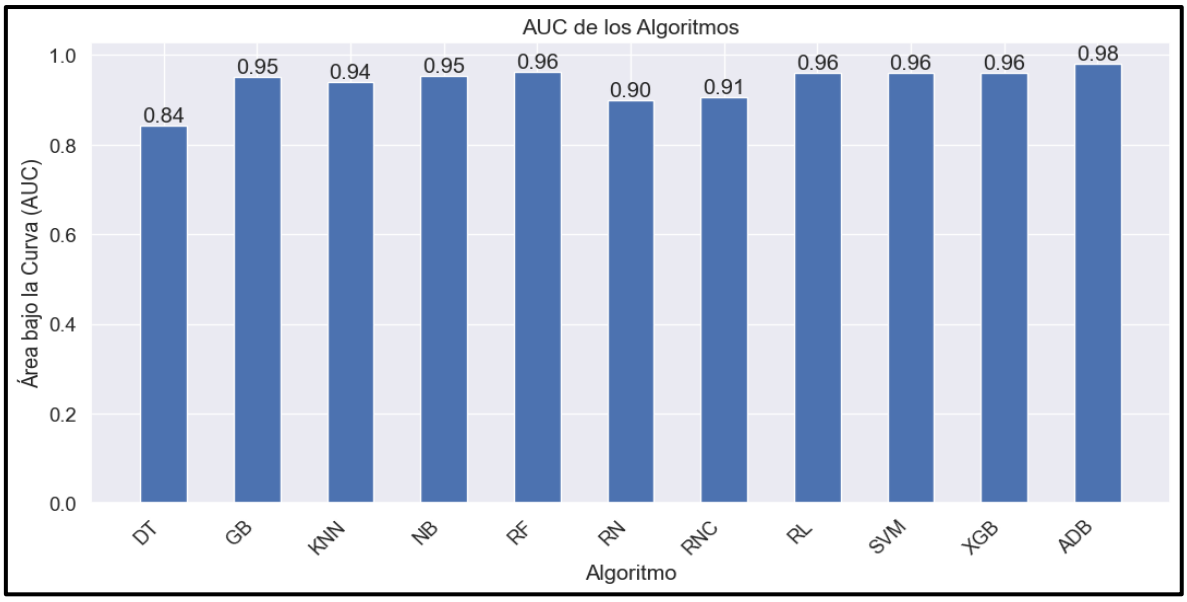
Figura Nº 37: AUC de los modelos Stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que el modelo stacking 5 obtuvo un 0.968761 de AUC, siendo así el mejor resultado en función a la métrica AUC.

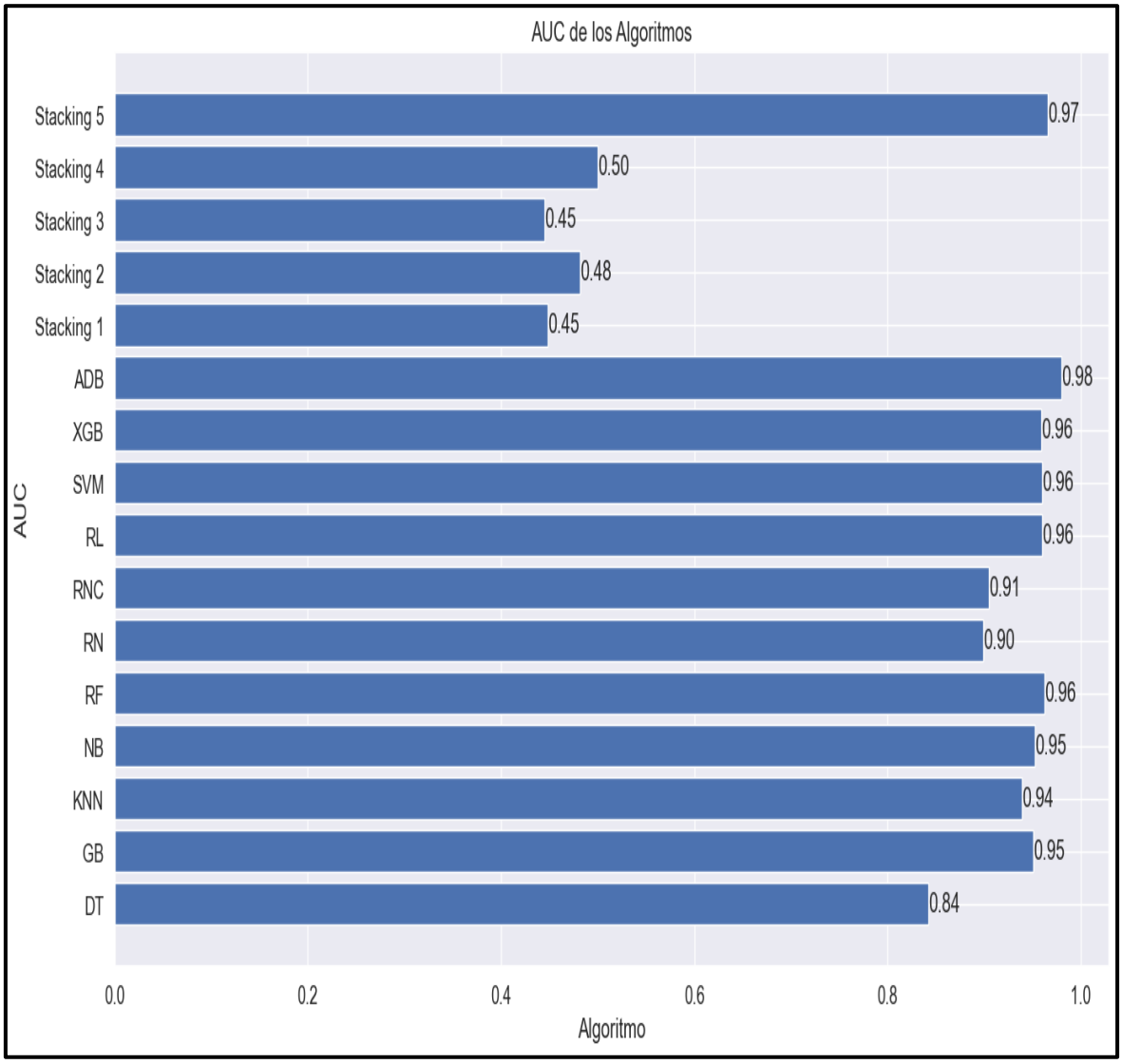
Figura Nº 38: AUC de los algoritmos



Fuente: Elaboración propia

Interpretación: Se visualiza que el algoritmo que tuvo mejor resultado de AUC es AdaBoost (0.98) y los modelos Random Forest, Regresión logística, SVM y XGBoost obtuvieron un 0.96 de AUC.

Figura Nº 39: AUC de los algoritmos y modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que el algoritmo que tuvo mejor resultado de AUC es AdaBoost (0.98) y los modelos Random Forest, Regresión logística, SVM y XGBoost obtuvieron un 0.96 de AUC. Realizando una comparación con los modelos stacking, el modelo stacking 5 obtuvo mejores resultados frente a la métrica de AUC, teniendo resultados cercanos a los mejores modelos propuestos.

Objetivo 6: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la especificidad.

- **Decision Tree**

Tabla Nº 107: Calculo de especificidad con el algoritmo - DT

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(44/44+7) * 100	86.27%
1	(39/39+4) * 100	90.69%
2	(30/30+0) * 100	100%
Total		93.70%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 92.06% usando el algoritmo Decision Tree.

- **Gradient Boosting**

Tabla Nº 108: Calculo de especificidad del algoritmo - GB

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(46/46+5) * 100	90.20%
1	(40/40+3) * 100	93.02%
2	(30/30+0) * 100	100%
Total		94.35%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 94.35% usando el algoritmo Gradient Boosting.

- **KNN**

Tabla Nº 109: Calculo de especificidad con el algoritmo - KNN

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(47/47+4) * 100	92.15%
1	(39/39+4) * 100	90.69%
2	(29/29+1) * 100	96.66%
Total		92.71%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 92.71% usando el algoritmo KNN.

- **Naive Bayes**

Tabla Nº 110: Calculo de especificidad con el algoritmo - NB

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(46/46+5) * 100	90.20%
1	(40/40+3) * 100	93.02%
2	(29/29+1) * 100	96.67%
Total		92.81%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 92.81% usando el algoritmo Naive Bayes

- **Nearest Centroid**

Tabla Nº 111: Calculo de especificidad con el algoritmo - NC

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(54/54+12) * 100	81.82%
1	(52/52+11) * 100	82.54%
2	(50/50+1) * 100	98.04%
Total		86.19%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una especificidad de 86.19% usando el algoritmo Nearest Centroid.

- **Random Forest**

Tabla Nº 112: Calculo de especificidad con el algoritmo - RF

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(46/46+5) * 100	90.196%
1	(41/41+2) * 100	95.35%
2	(30/30+0) * 100	100%
Total		95.14%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 95.14% usando el algoritmo Random Forest.

- **Redes Neuronales**

Tabla Nº 113: Calculo de especificidad con el algoritmo - RN

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(36/36+14) * 100	98%
1	(43/43+1) * 100	75%
2	(28/28+2) * 100	93.3%
Total		85.26%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 85.26% usando el algoritmo Redes Neuronales.

- **Redes neuronales convolucionales**

Tabla Nº 114: Calculo de especificidad con el algoritmo - RNC

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(42/42+8) * 100	72%
1	(38/38+6) * 100	97.73%
2	(28/28+2) * 100	93.3%
Total		85.74%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 85.74% usando el algoritmo redes neuronales convolucionales Red neuronal convolucional.

- **Regresión Logística**

Tabla Nº 115: Calculo de especificidad con el algoritmo - RL

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(57/57+9) * 100	86.36%
1	(62/62+1) * 100	98.41%
2	(50/50+1) * 100	98.04%
Total		94.06%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables y balanceo de datos permite predecir enfermedades de la columna vertebral con una especificidad de 94.06% usando el algoritmo Regresión Logística.

- **Support Vector Machine**

Tabla Nº 116: Calculo de especificidad con el algoritmo SVM

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(45/45+5) * 100	90%
1	(42/42+2) * 100	95.45%
2	(29/29+1) * 100	96.67%
Total		93.69%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 93.69% usando el algoritmo Support Vector Machine

- **XGBoost**

Tabla Nº 117: Calculo de especificidad con el algoritmo - XGB

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(47/47+5) * 100	90.38%
1	(41/41+2) * 100	95.35%
2	(30/30+0) * 100	100%
Total		95.14%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 95.14% usando el algoritmo XGBoost.

- **AdaBoost**

Tabla Nº 118: Calculo de especificidad con el algoritmo - ADA

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(61/61+5) * 100	92.42%
1	(61/61+2) * 100	96.83%
2	(50/50+1) * 100	98.04%
Total		95.60%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 95.60% usando el algoritmo AdaBoost.

- **Stacking 1**

Tabla Nº 119: Calculo de especificidad con el modelo Stacking 1

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(47/47+4) * 100	92.15%
1	(40/40+3) * 100	93.02%
2	(30/30+0) * 100	100%
Total		95.01%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 95.01% usando el algoritmo Stacking 1.

- **Stacking 2**

Tabla Nº 120: Calculo de especificidad con el modelo Stacking 2

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(48/48+3) * 100	94.11%
1	(39/39+4) * 100	90.69%
2	(30/30+0) * 100	100%
Total		94.86%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 94.86% usando el algoritmo Stacking 2.

- **Stacking 3**

Tabla Nº 121: Calculo de especificidad con el modelo Stacking 3

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(47/47+4) * 100	92.15%
1	(40/40+3) * 100	93.02%
2	(30/30+0) * 100	100%
Total		95.01%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 95.01% usando el algoritmo Stacking 3.

- **Stacking 4**

Tabla Nº 122: Calculo de especificidad con el modelo Stacking 4

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(47/47+4) * 100	92.15%
1	(40/40+3) * 100	93.02%
2	(30/30+0) * 100	100%
Total		95.01%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una especificidad de 95.01% usando el algoritmo Stacking 4

- **Stacking 5**

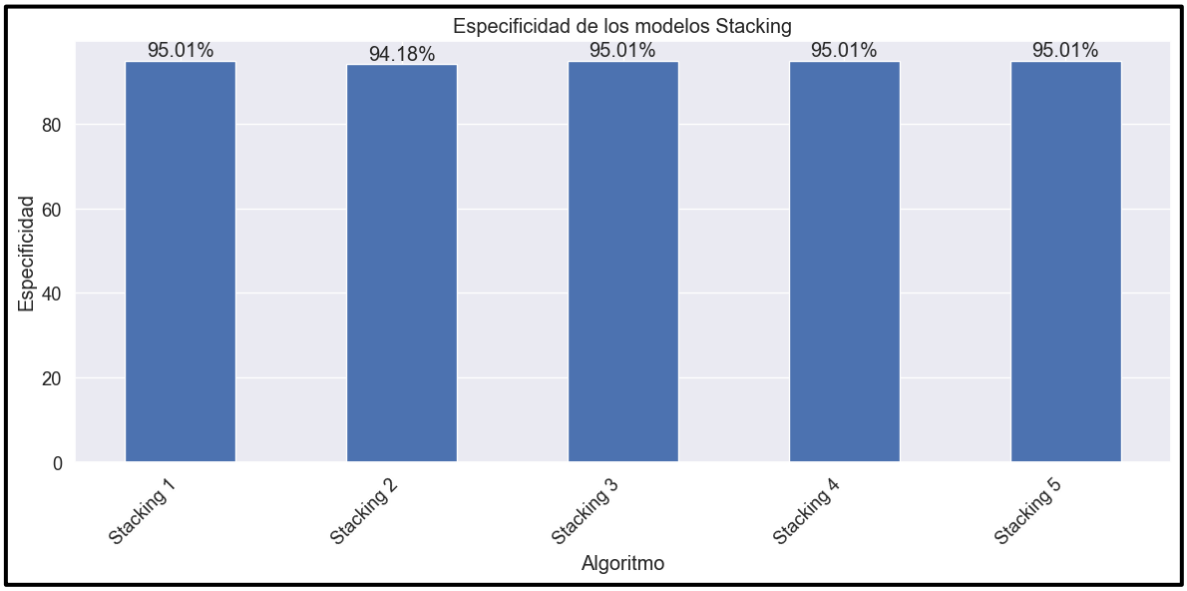
Tabla Nº 123: Calculo de especificidad con el modelo Stacking 5

Clases	Especificidad (TN/TN+FP) * 100	Resultado
0	(47/47+4) * 100	92.15%
1	(40/40+3) * 100	93.02%
2	(30/30+0) * 100	100%
Total		95.01%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con una exactitud de 95.01% usando el algoritmo Stacking 5.

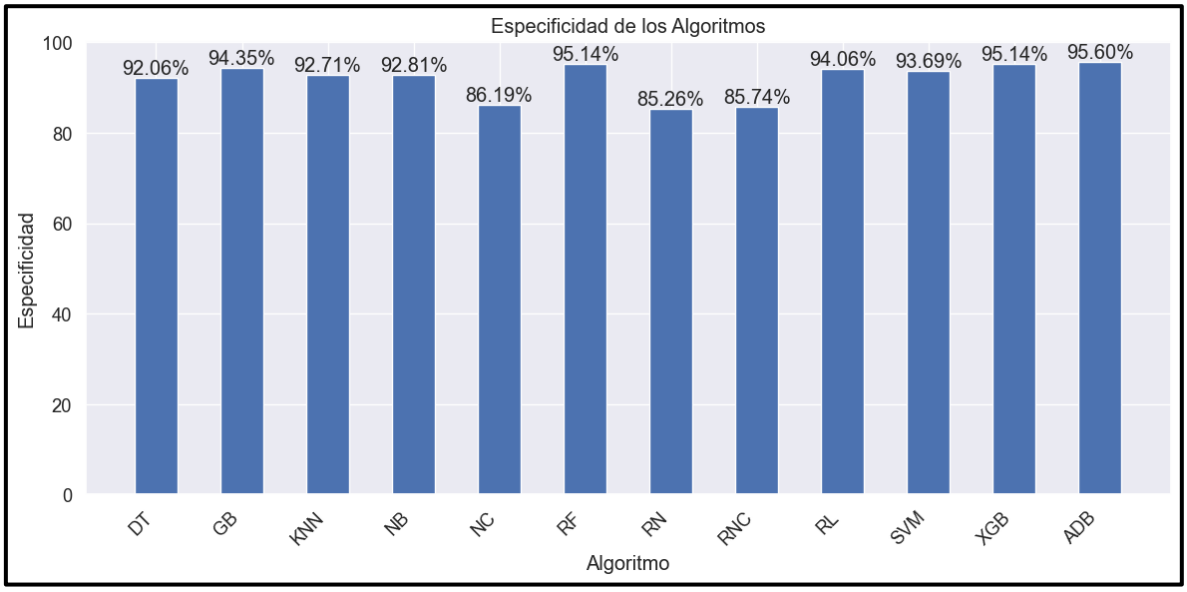
Figura Nº 40: Especificidad de los modelos Stacking



Fuente Elaboración propia

Interpretación: Se visualiza que los modelos stacking 1, stacking 3, stacking 4y stacking 5 obtuvieron una especificidad de 95.01%, por otor lado el stacking 2 obtuvo una especificidad de 94.18%.

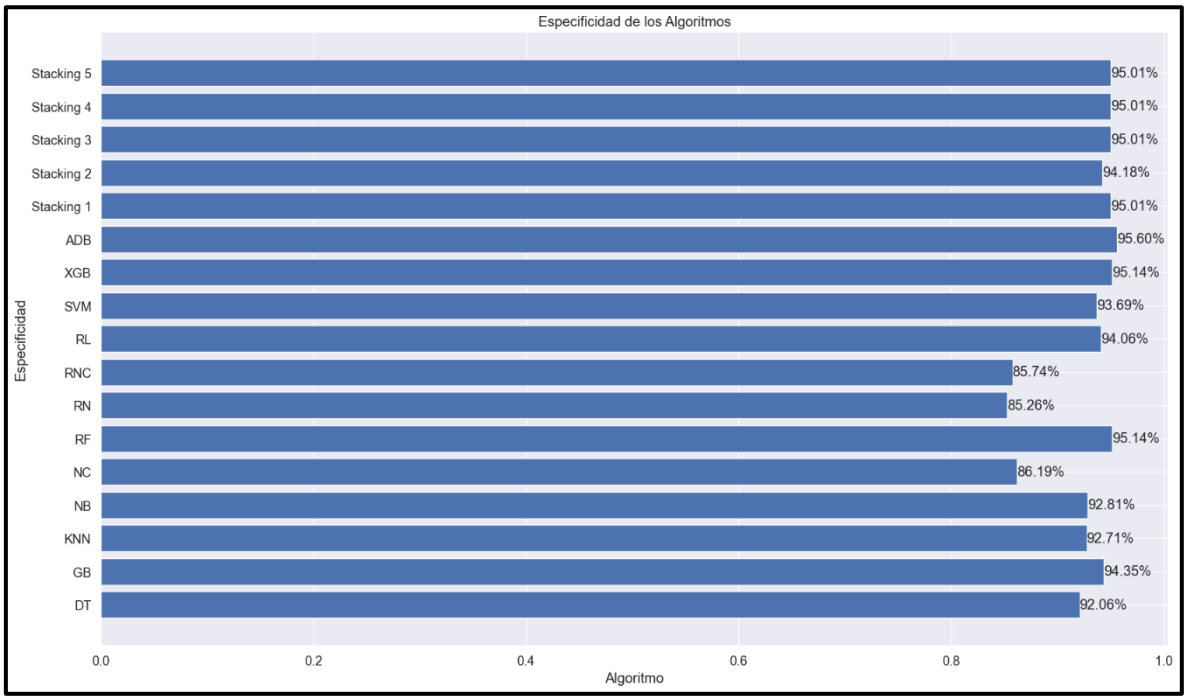
Figura Nº 41: Especificidad de los algoritmos



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que tuvieron una mejor especificidad son AdaBoost (95.60%) y los modelos XGBoost y Random Forest obtuvieron el mismo porcentaje de especificidad de 95.14%.

Figura Nº 42: Especificidad de los algoritmos y modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que el algoritmo que tuvo una mejor especificidad es AdaBoost (95.60%).

Objetivo 7: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la MCC.

- **Decision Tree**

Tabla Nº 124: Calculo de MCC con el algoritmo - DT

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×44)−(7×2) /√(9+7)(9+2)(44+7)(44+2)	0.61
1	((13×39)−(4×6)/√(13+4)(13+6)(39+4)(39+6))	0.90
2	((29×30)−(0×3)/√ (29+0)(29+3)(30+0)(30+3))	0.907
Total		76.12%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 76.12% usando el algoritmo Decision Tree.

- **Gradient Boosting**

Tabla Nº 125: Calculo de MCC con el algoritmo - GB

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(8×46)−(5×3)/√(8+5)(8+3)(46+5)(46+3)	0.59
1	(15×40)−(3×4)/√(15+3)(15+4)(40+3)(40+4)	0.73
2	(31×30)−(0×1)/√(31+0)(31+1)(30+2)(30+1)	0.96
Total		82.85%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 82.85% usando el algoritmo Gradient Boosting.

- **KNN**

Tabla Nº 126: Calculo de MCC con el algoritmo - KNN

Clases	MCC $((TP\times TN)-(FP\times FN)/\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)})*100$	Resultado
0	$(9\times47)-(4\times2)/\sqrt{(9+4)(9+2)(47+4)(47+2)}$	0.694
1	$(15\times39)-(4\times4)/\sqrt{(15+4)(15+4)(39+4)(39+4)}$	0.694
2	$(29\times29)-(1\times3)/\sqrt{(29+1)(29+3)(29+1)(29+3)}$	0.872
Total		78.71%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 78.71% usando el algoritmo KNN.

- **Naive Bayes**

Tabla Nº 127: Calculo de MCC con el algoritmo - NB

Clases	MCC $((TP\times TN)-(FP\times FN)/\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)})*100$	Resultado
0	$(10\times46)-(5\times1)/\sqrt{(10+5)(10+1)(46+5)(46+1)}$	0.723
1	$(13\times40)-(4\times6)/\sqrt{(13+4)(13+6)(40+4)(40+6)}$	0.647
2	$(30\times29)-(1\times2)/\sqrt{(30+1)(30+2)(29+1)(29+2)}$	0.903
Total		79.32%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 79.32% usando el algoritmo Naive Bayes.

- **Nearest Centroid**

Tabla Nº 128: Calculo de MCC con el algoritmo - NC

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(20×54)−(12×4) /√ (20+12)(20+4)(54+12)(54+4)	0.601
1	(16×52)−(11×11) /√ (16+11)(16+11)(52+11)(52+11)	0.417
2	(30×50)−(1×9) /√ (30+1)(30+9)(50+1)(50+9)	0.781
Total		62.46%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 62.46% usando el algoritmo Naive Bayes.

- **Random Forest**

Tabla Nº 129: Calculo de MCC con el algoritmo - RF

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×46)−(5×2)(9+5)(9+2)(46+5)(46+2)	0.657
1	(15×41)−(2×4) (15+2)(15+4)(41+2)(41+4)	0.767
2	(31×30)−(0×1)(31+0)(31+1)(30+0)(30+1)	0.968
Total		85.18%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 85.18% usando el algoritmo Random Forest.

- **Red Neuronal**

Tabla Nº 130: Calculo de MCC con el algoritmo - RN

Clases	MCC $((TP\times TN)-(FP\times FN)/\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)})*100$	Resultado
0	(10×36)−(14×2)(0+1)(0+1)(49+1)(49+1)	0.448
1	(3×43)−(1×15)(3+1)(3+15)(43+1)(43+15)	0.265
2	(32×28)−(2×0)(32+2)(32+0)(28+2)(28+0)	0.937
Total		64.78%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 64.78%usando el algoritmo Red Neuronal.

- **Redes neuronales convolucionales**

Tabla Nº 131: Calculo de MCC con el algoritmo - RNC

Clases	MCC $((TP\times TN)-(FP\times FN)/\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)})*100$	Resultado
0	(5×42)−(8×7)(7+5)(7+5)(45+5)(45+5)	0.249
1	(9×38)−(6×9)(9+6)(9+9)(38+6)(38+9)	0.385
2	(32×28)−(2×0)(32+2)(32+0)(28+2)(28+0)	0.937
Total		64.38%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 64.38% usando el algoritmo redes neuronales convolucionales.

- **Regresión Logística**

Tabla Nº 132: Calculo de MCC con el algoritmo - RL

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(23×57)−(9×1) /√ (23+9)(23+1)(57+9)(57+1)	0.759
1	(19×62)−(1×8) /√ (16+1)(16+8)(62+1)(62+8)	0.758
2	(37×50)−(1×2) /√ (37+1)(37+2)(50+1)(50+2)	0.932
Total		83.39%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 83.39 % usando el algoritmo Regresión Logística.

- **Support Vector Machine (SVM)**

Tabla Nº 133: Calculo de MCC con el algoritmo - SVM

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	((9×45)−(5×3)/√(9+5)(9+3)(45+5)(45+3))*100	0.614
1	((14×42)−(2×4)/√(14+2)(14+4)(42+2)(42+4))	0.759
2	((31×29)−(1×1)/√(31+1)(31+1)(29+1)(29+1))	0.935
Total		82.22%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 82.22% usando el algoritmo Support Vector Machine.

- **XGBoost**

Tabla Nº 134: Calculo de MCC con el algoritmo - XGB

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×47)−(5×2) /√ (9+5)(9+2)(47+5)(47+2) *100	0.657
1	(15×41)−(2×4) /√ (15+2)(15+4)(41+2)(41+4) *100	0.767
2	(31×30)−(0×1) /√ (31+0)(31+1)(30+0)(30+1) *100	0.969
Total		85.18%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 85.18% usando el algoritmo XGBoost.

- **AdaBoost**

Tabla Nº 135: Calculo de MCC con el algoritmo - ADA

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(23×61)−(5×1) /√ (23+5)(23+1)(61+5)(61+1) *100	0.843
1	(21×61)−(2×6) /√ (21+2)(21+6)(61+2)(61+6) *100	0.783
2	(38×50)−(1×1) /√ (38+1)(38+1)(50+1)(50+1) *100	0.954
Total		87.37%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 87.37% usando el algoritmo AdaBoost.

- **Stacking 1**

Tabla Nº 136: Calculo de MCC con el modelo Stacking 1

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×47)−(4×2)(9+4)(9+2)(47+4)(47+2)	0.694
1	(15×40)−(3×4)(15+3)(15+4)(40+3)(40+4)	0.730
2	(31×30)−(0×1)(31+0)(31+1)(30+0)(30+1)	0.968
Total		84.69%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 84.69% usando el algoritmo Stacking 1.

- **Stacking 2**

Tabla Nº 137: Calculo de MCC con el modelo Stacking 2

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×48)−(3×2)(9+3)(9+2)(48+3)(48+2)	0.694
1	(16×39)−(4×3)(16+4)(16+3)(39+4)(39+3)	0.696
2	(30×30)−(0×2)(30+0)(30+2)(30+0)(30+2)	0.9375
Total		82.05%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 82.05% usando el algoritmo Stacking 2.

- **Stacking 3**

Tabla Nº 138: Calculo de MCC con el modelo Stacking 3

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×47)−(4×2)(9+4)(9+2)(47+4)(47+2)	0.694
1	(15×40)−(3×4)(15+3)(15+4)(40+3)(40+4)	0.730
2	(31×30)−(0×1)(31+0)(31+1)(30+0)(30+1)	0.968
Total		84.69%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 84.69% usando el algoritmo Stacking 3.

- **Stacking 4**

Tabla Nº 139: Calculo de MCC con el modelo Stacking 4

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×47)−(4×2)(9+4)(9+2)(47+4)(47+2)	0.694
1	(15×40)−(3×4)(15+3)(15+4)(40+3)(40+4)	0.730
2	(31×30)−(0×1)(31+0)(31+1)(30+0)(30+1)	0.968
Total		84.69%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 84.69% usando el algoritmo Stacking 4.

- **Stacking 5**

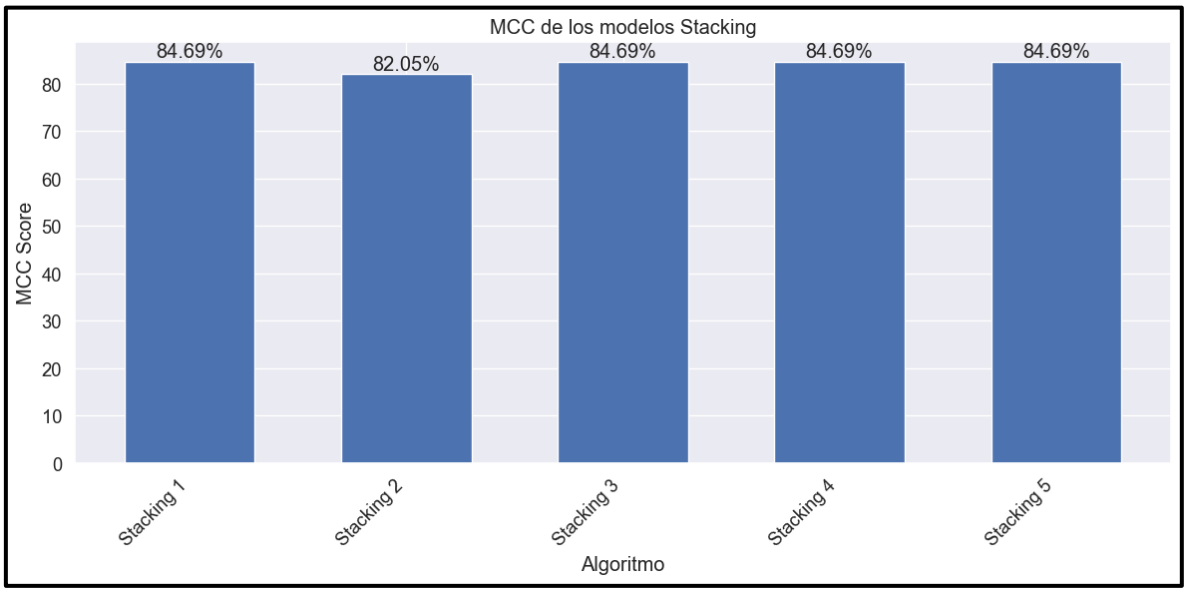
Tabla Nº 140: Calculo de MCC con el modelo Stacking 5

Clases	MCC ((TP×TN)−(FP×FN)/√(TP+FP)(TP+FN)(TN+FP)(TN+FN))*100	Resultado
0	(9×47)−(4×2)(9+4)(9+2)(47+4)(47+2)	0.694
1	(15×40)−(3×4)(15+3)(15+4)(40+3)(40+4)	0.730
2	(31×30)−(0×1)(31+0)(31+1)(30+0)(30+1)	0.968
Total		84.69%

Fuente: Elaboración propia

Interpretación: El desarrollo de un Sistema Inteligente con Machine Learning, basado en selección de variables permite predecir enfermedades de la columna vertebral con un valor MCC de 84.69% usando el algoritmo Stacking 5.

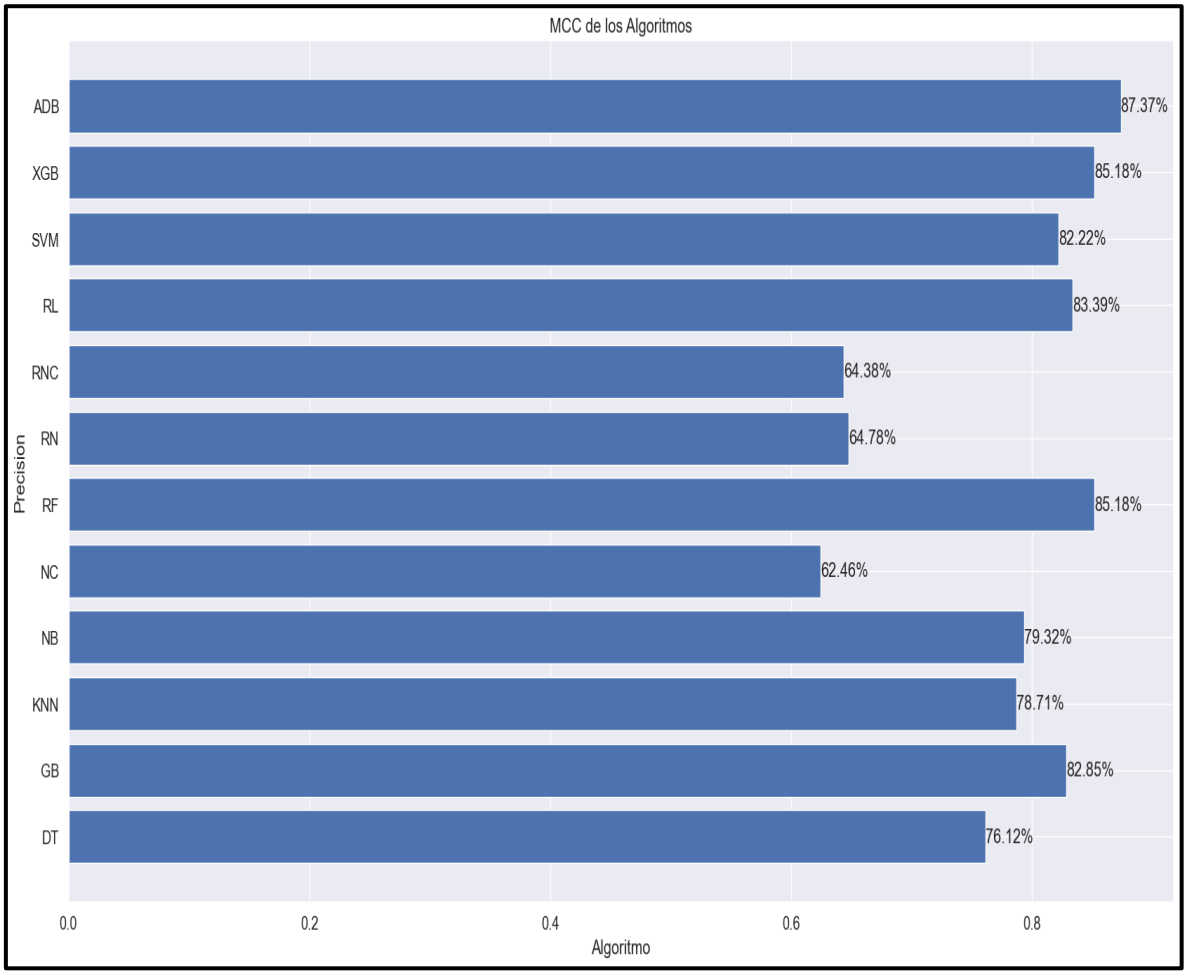
Figura Nº 43: MCC de los modelos Stacking



Fuente: Elaboración propia

Interpretación: Se observa que los modelos stacking 1, stacking 3, stacking 4 y stacking 5 obtuvieron el mismo porcentaje de 81,77%. Por otro lado, el modelo stacking 2 obtuvo 79.30% en función a la métrica MCC.

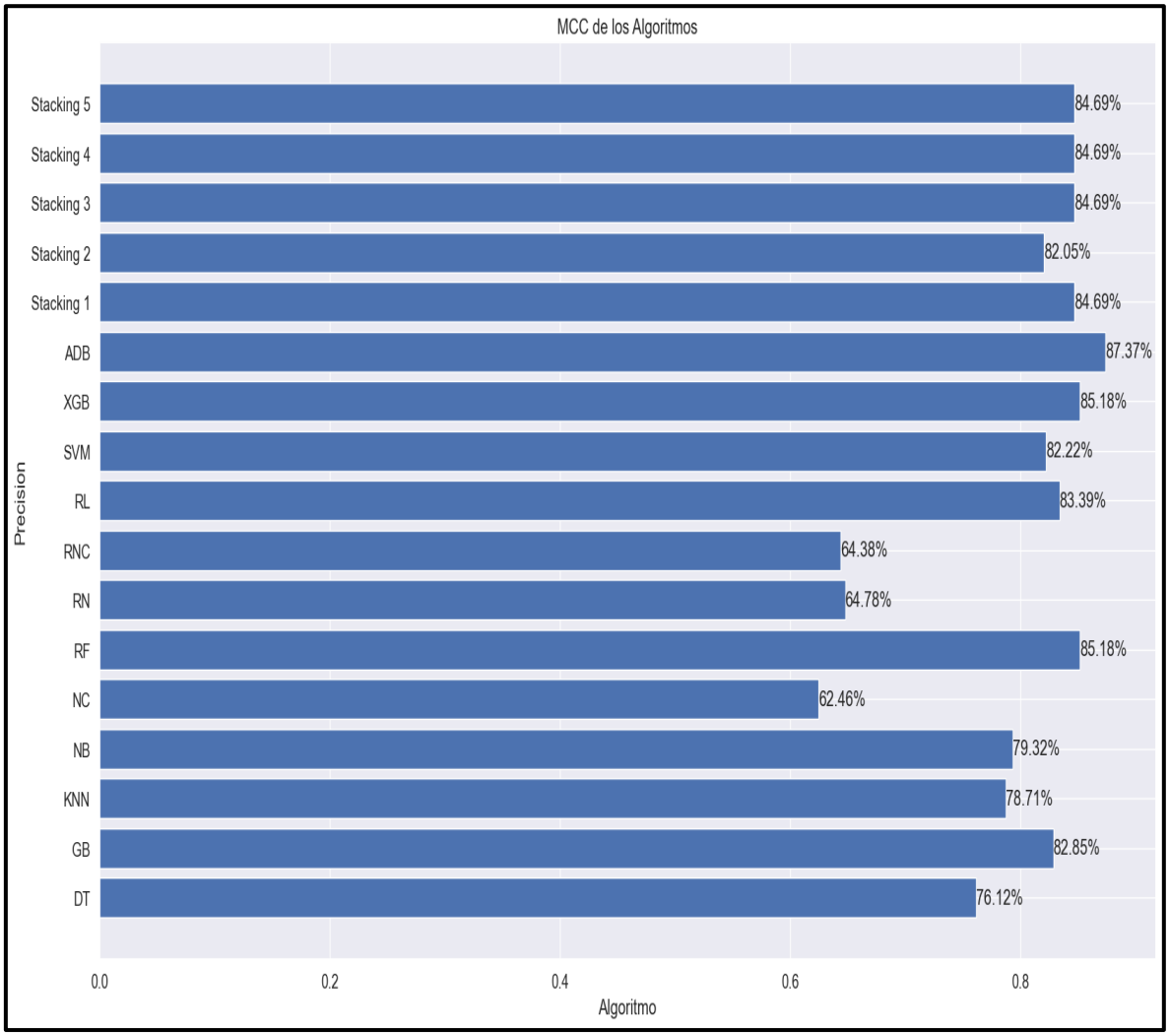
Figura Nº 44: MCC de los algoritmos



Fuente: Elaboración propia

Interpretación: Se observa que los algoritmos que tuvieron un mejor porcentaje en función a la métrica MCC es AdaBoost (87.37%). Por otro lado, los modelos XGBoost y Random Forest obtuvieron el mismo porcentaje de 85.18%.

Figura Nº 45: MCC de los algoritmos y modelos stacking



Fuente: Elaboración propia

Interpretación: Se visualiza que los algoritmos que tuvieron un mejor porcentaje en función a la métrica MCC es AdaBoost (87.37%). Por otro lado, los modelos XGBoost y Random Forest obtuvieron el mismo porcentaje de 85.18%. Realizando una comparación de los modelos stacking y los algoritmos planteados para crear los modelos stacking, los algoritmos AdaBoost, XGBoost y Random Forest superaron a los modelos stacking propuestos.

Objetivo 8: El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la pérdida.

- **Redes neuronales**

Valores

n = 310 (número total de muestras)

m = 3 (número de clases)

Yij = (etiquetas reales)

Figura Nº 46: Etiquetas reales - RN

```
Etiquetas reales de las primeras 10 filas del conjunto de prueba:
['Normal' 'Hernia' 'Hernia' 'Spondylolisthesis' 'Hernia'
 'Spondylolisthesis' 'Spondylolisthesis' 'Spondylolisthesis' 'Normal'
 'Hernia']
```

Fuente: Elaboración propia

^*ij* = (probabilidad predicha por el modelo)

Figura Nº 47: Probabilidades predichas - RN

```
Probabilidades Predichas (y_pred_probs):
[[8.8848037e-01 1.1151929e-01 3.6014123e-07]
 [1.7087635e-01 8.2791656e-01 1.2070666e-03]
 [8.1536674e-01 1.8460338e-01 2.9880774e-05]
 [1.0706436e-08 1.0882503e-09 1.0000000e+00]
 [9.8312885e-01 1.6835239e-02 3.5875393e-05]
 [5.4454372e-08 6.3313287e-08 9.9999988e-01]
 [3.9856044e-14 1.1672741e-14 1.0000000e+00]
 [9.9146753e-11 1.1654707e-09 1.0000000e+00]
 [8.5588777e-01 8.2783177e-02 6.1329044e-02]
 [9.6275008e-01 3.6934234e-02 3.1573439e-04]]
```

Fuente: Elaboración propia

Tabla Nº 141: Calculo de Cross entropy del algoritmo - RN

Loss **CrossEntropy=−*n*1∑*i*=1*n* ∑*j*=1*myij*log(*y*^*ij*)**	**Resultado**
Total	0.86

Fuente: Elaboración propia

Figura Nº 48: Entrenamiento del algoritmo - RN

```
Epoch 25/30
5/5 [==============================] - 0s 8ms/step - loss: 2.0923 - accuracy: 0.6148 - val_loss: 0.7674 - val_accuracy: 0.6056
Epoch 26/30
5/5 [==============================] - 0s 9ms/step - loss: 2.3877 - accuracy: 0.6325 - val_loss: 0.8793 - val_accuracy: 0.5634
Epoch 27/30
5/5 [==============================] - 0s 9ms/step - loss: 1.9861 - accuracy: 0.6643 - val_loss: 0.9811 - val_accuracy: 0.5352
Epoch 28/30
5/5 [==============================] - 0s 10ms/step - loss: 2.4494 - accuracy: 0.6466 - val_loss: 1.0194 - val_accuracy: 0.5352
Epoch 29/30
5/5 [==============================] - 0s 8ms/step - loss: 2.1228 - accuracy: 0.6572 - val_loss: 0.9388 - val_accuracy: 0.5493
Epoch 30/30
5/5 [==============================] - 0s 9ms/step - loss: 2.3050 - accuracy: 0.5936 - val_loss: 0.8035 - val_accuracy: 0.5775
2/2 [==============================] - 0s 3ms/step
```

Fuente: Elaboración propia

Interpretación: Mediante la aplicación del algoritmo de Redes Neuronales se obtuvo una pérdida de 2.30. Por otro lado, usando la fórmula de entropía cruzada categórica se obtuvo un 0.60 de pérdida.

- **Redes neuronales convolucionales**

Valores

n = 310 (número total de muestras)

m = 3 (número de clases)

Yij = (etiquetas reales)

Figura Nº 49: Etiquetas reales - RNC

```
Etiquetas reales de las primeras 10 filas del conjunto de prueba:
['Normal' 'Hernia' 'Hernia' 'Spondylolisthesis' 'Hernia'
 'Spondylolisthesis' 'Spondylolisthesis' 'Spondylolisthesis' 'Normal'
 'Hernia']
```

Fuente: Elaboración propia

^ij = (probabilidad predicha por el modelo)

Figura Nº 50: Probabilidades predichas - RNC

```
Probabilidades Predichas (y_pred_probs):
[[6.5050918e-01 3.4948999e-01 8.1698556e-07]
 [7.0880773e-03 9.7011232e-01 2.2799613e-02]
 [4.4513354e-01 5.5416840e-01 6.9809117e-04]
 [1.5971768e-10 4.6966719e-11 1.0000000e+00]
 [9.6605474e-01 3.3886831e-02 5.8344704e-05]
 [6.8644049e-08 9.7915006e-08 9.9999988e-01]
 [5.0629610e-16 3.4512569e-16 1.0000000e+00]
 [1.4351031e-11 2.2169190e-10 1.0000000e+00]
 [4.6760488e-01 3.4730545e-01 1.8508969e-01]
 [9.0613049e-01 9.3098313e-02 7.7123143e-04]]
```

Fuente: Elaboración propia

Tabla Nº 142: Calculo de Cross entropy del algoritmo - RNC

Loss CrossEntropy=−*n*1∑*i*=1*n* ∑*j*=1*myij*log(*y*^*ij*)	Resultado
Total	0.60

Fuente: Elaboración propia

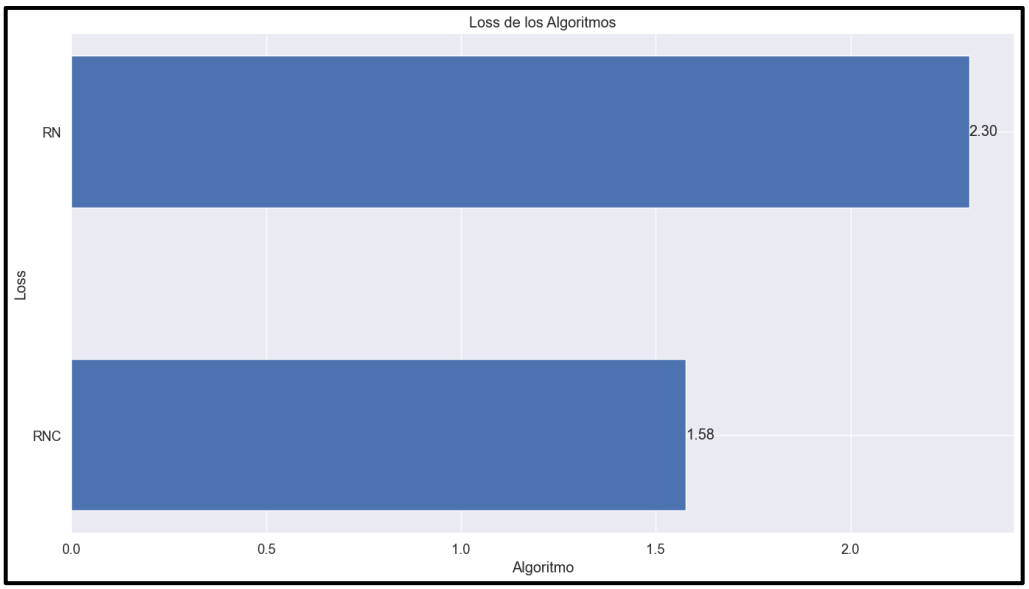
Figura Nº 51: Entrenamiento del algoritmo - RNC

```
Epoch 24/30
5/5 [==============================] - 0s 15ms/step - loss: 2.4175 - accuracy: 0.6113 - val_loss: 1.0758 - val_accuracy: 0.6620
Epoch 25/30
5/5 [==============================] - 0s 8ms/step - loss: 2.4001 - accuracy: 0.6219 - val_loss: 1.0248 - val_accuracy: 0.6479
Epoch 26/30
5/5 [==============================] - 0s 11ms/step - loss: 2.3372 - accuracy: 0.6466 - val_loss: 0.9969 - val_accuracy: 0.6761
Epoch 27/30
5/5 [==============================] - 0s 8ms/step - loss: 1.8990 - accuracy: 0.6643 - val_loss: 1.0628 - val_accuracy: 0.6479
Epoch 28/30
5/5 [==============================] - 0s 9ms/step - loss: 2.3805 - accuracy: 0.6360 - val_loss: 1.1167 - val_accuracy: 0.6620
Epoch 29/30
5/5 [==============================] - 0s 14ms/step - loss: 1.7992 - accuracy: 0.6820 - val_loss: 1.1865 - val_accuracy: 0.6338
Epoch 30/30
5/5 [==============================] - 0s 9ms/step - loss: 1.5778 - accuracy: 0.6961 - val_loss: 1.1853 - val_accuracy: 0.6338
2/2 [==============================] - 0s 5ms/step
```

Fuente: Elaboración propia

Interpretación: Mediante la aplicación del algoritmo de Redes Neuronales Convolucionales se obtuvo una pérdida de 2.60. Por otro lado, usando la fórmula de entropía cruzada categórica se obtuvo un 1.288 de pérdida.

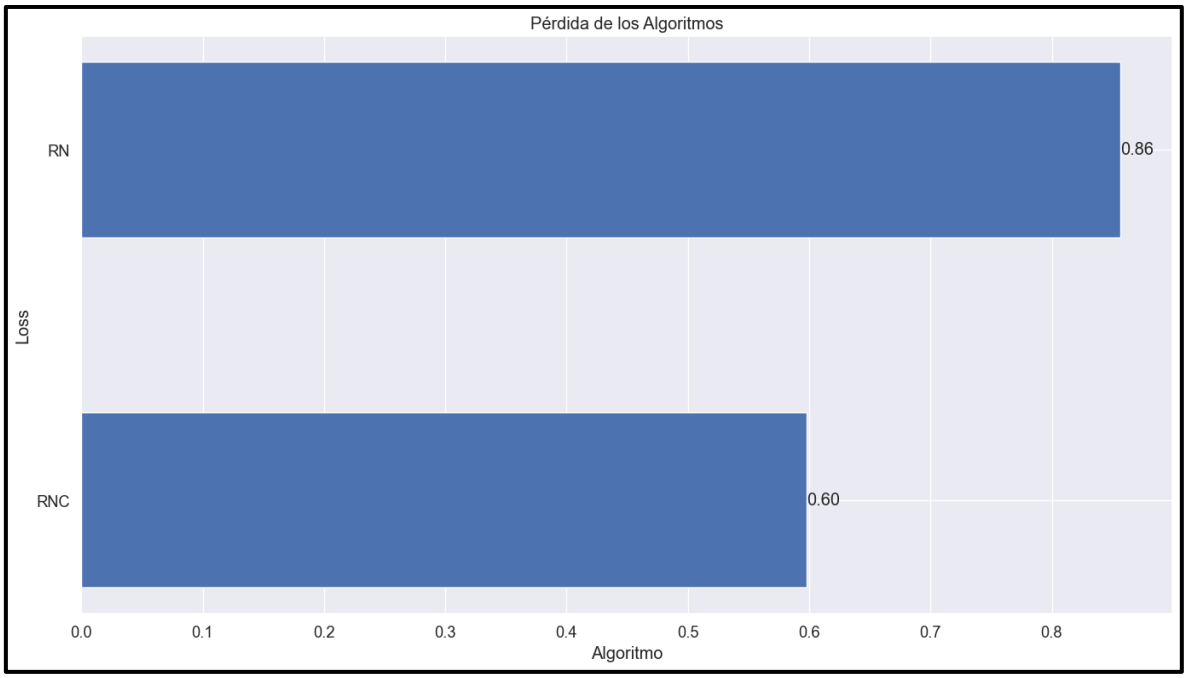
Figura Nº 51: Loss (pérdida) de los algoritmos RN y RNC



Fuente: Elaboración propia

Interpretación: Se visualiza que el algoritmo que tuvo mayor perdida es Redes Neuronales, con una pérdida de 2.30.

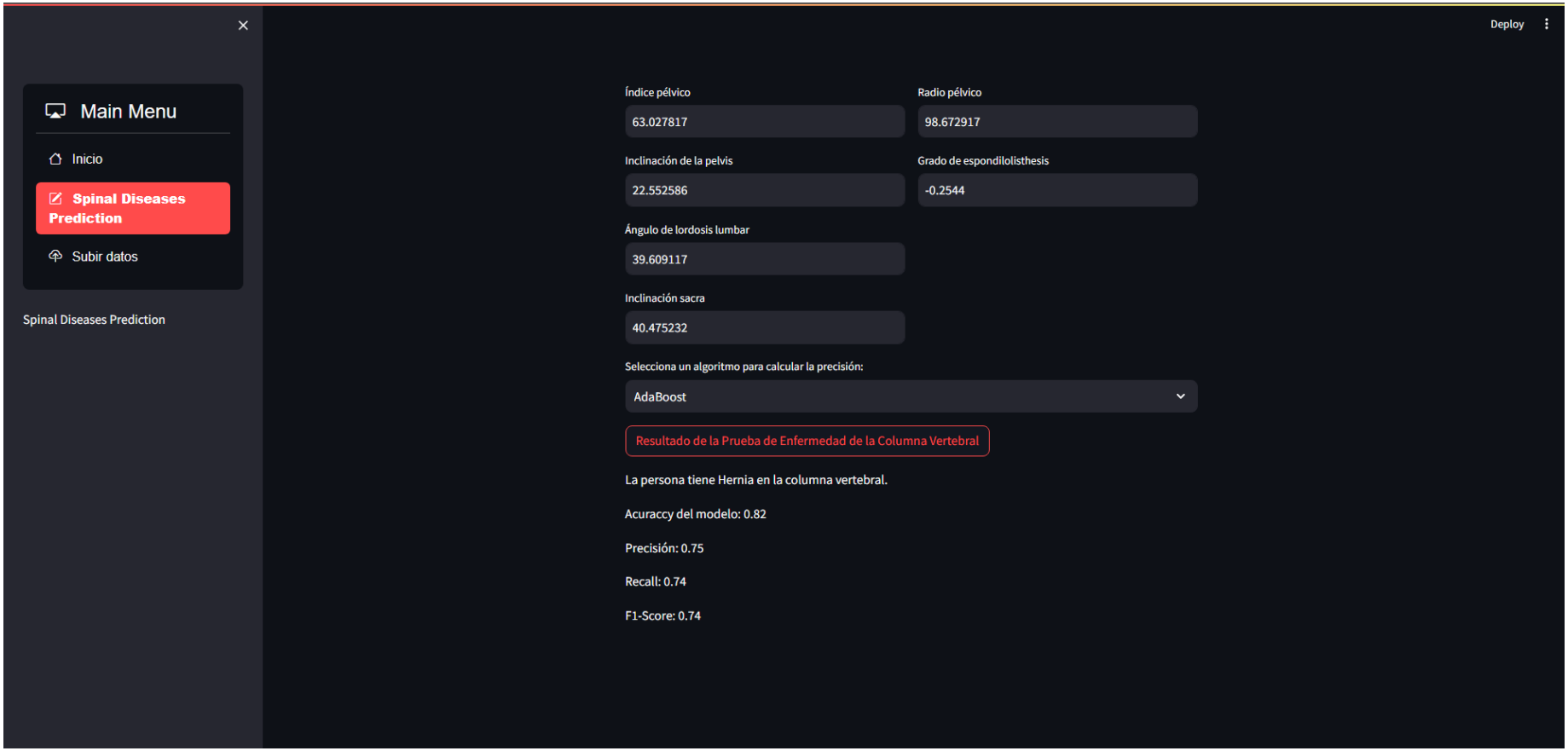
Figura Nº 52: Entropía Categórica Cruzada – RN y RNC



Fuente: Elaboración propia

Interpretación: Se visualiza que el algoritmo que tuvo mayor perdida en función de la fórmula de entropía categórica cruzada es Redes Neuronales (0.86).

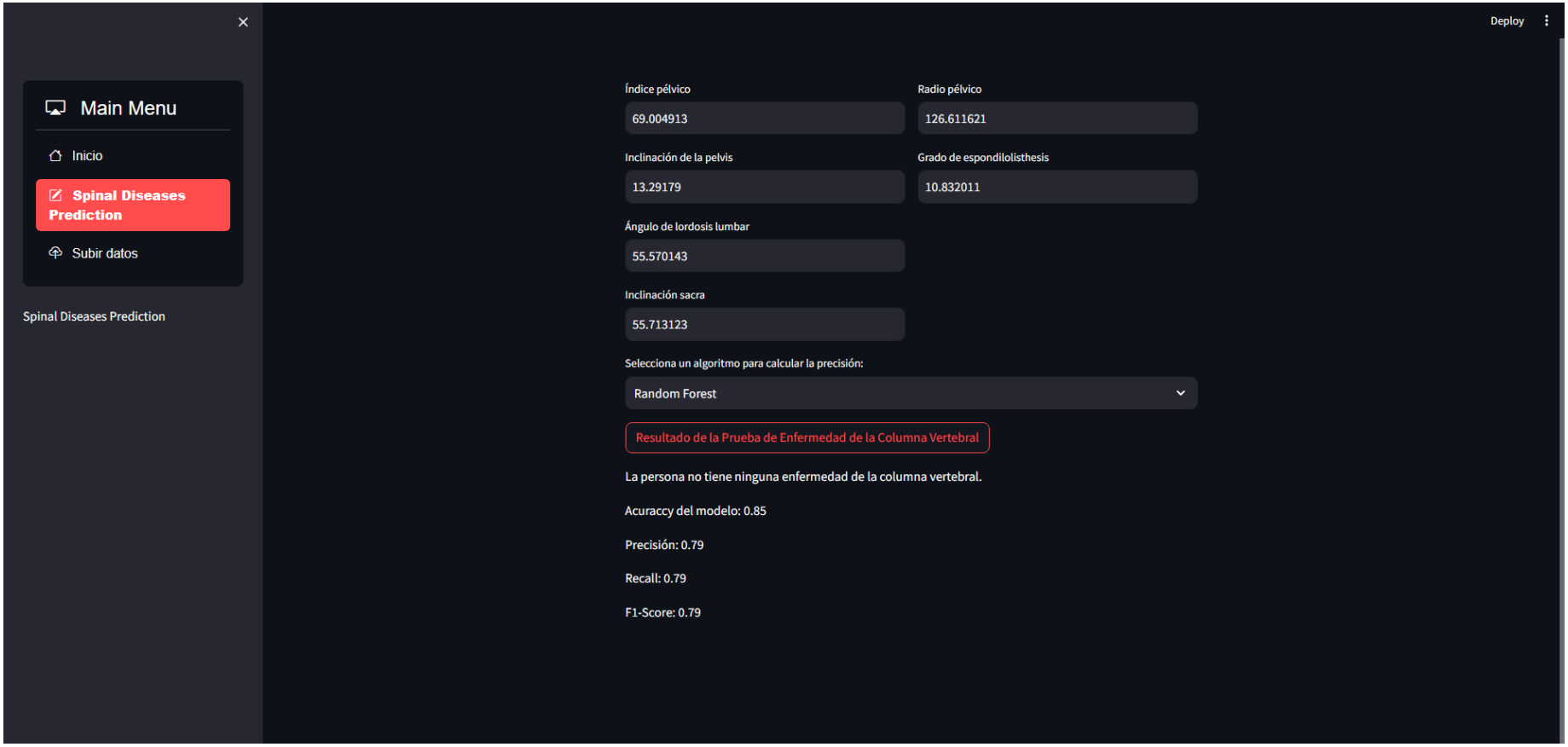
Figura N° 53: Resultado del Sistema Inteligente con el modelo AdaBoost



Fuente: Elaboración propia

Se visualiza los resultados que la persona tiene Hernia en la columna vertebral, también se observa las métricas que fueron usadas para la evaluación del modelo, en donde se obtuvo un 0.82 de exactitud.

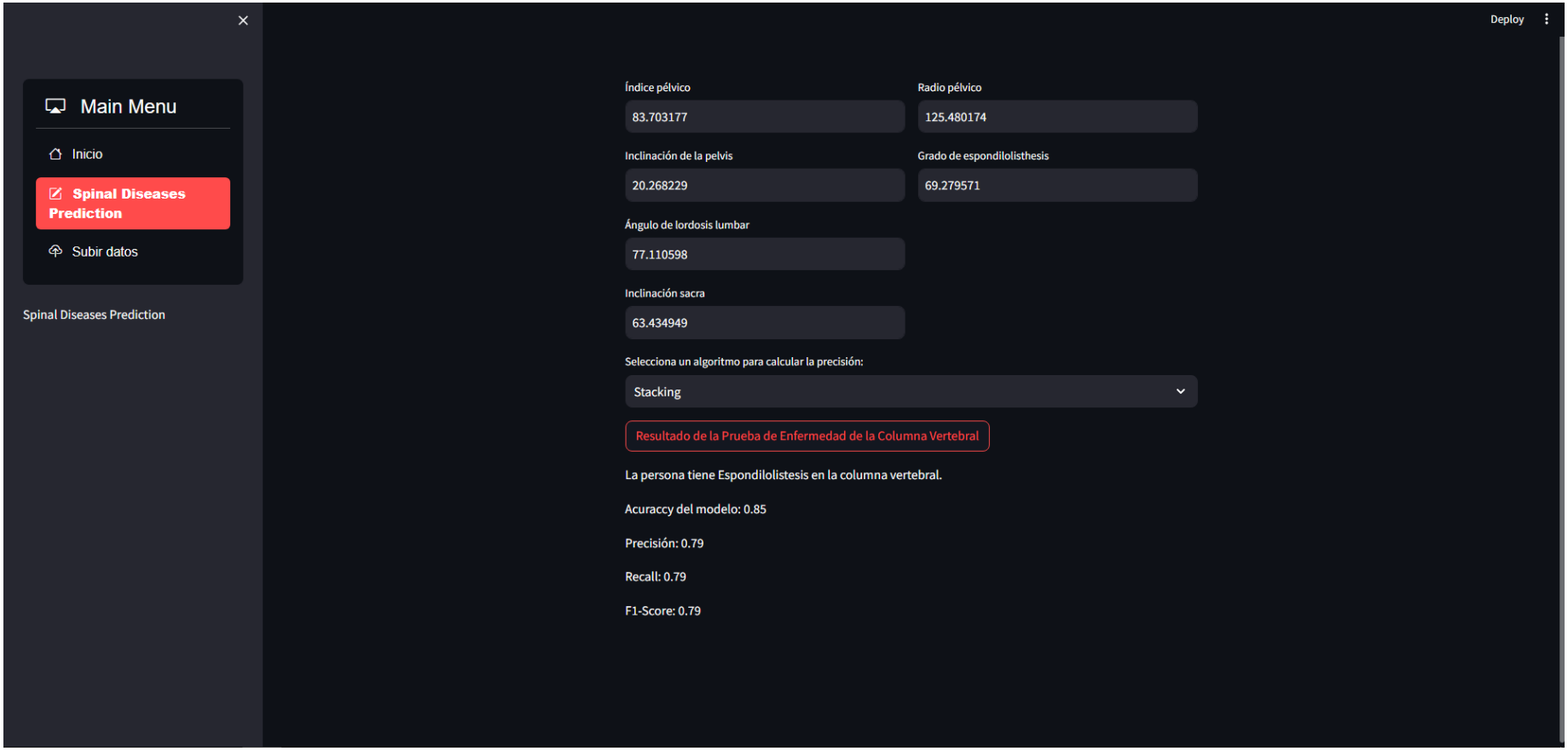
Figura N° 54: Resultado del Sistema Inteligente con el modelo Random Forest



Fuente: Elaboración propia

Se visualiza que mediante el uso del algoritmo Random Forest se obtuvo una exactitud de 0.85, en donde los datos de la persona que fue llenado, indica que no posee ninguna enfermedad de la columna vertebral.

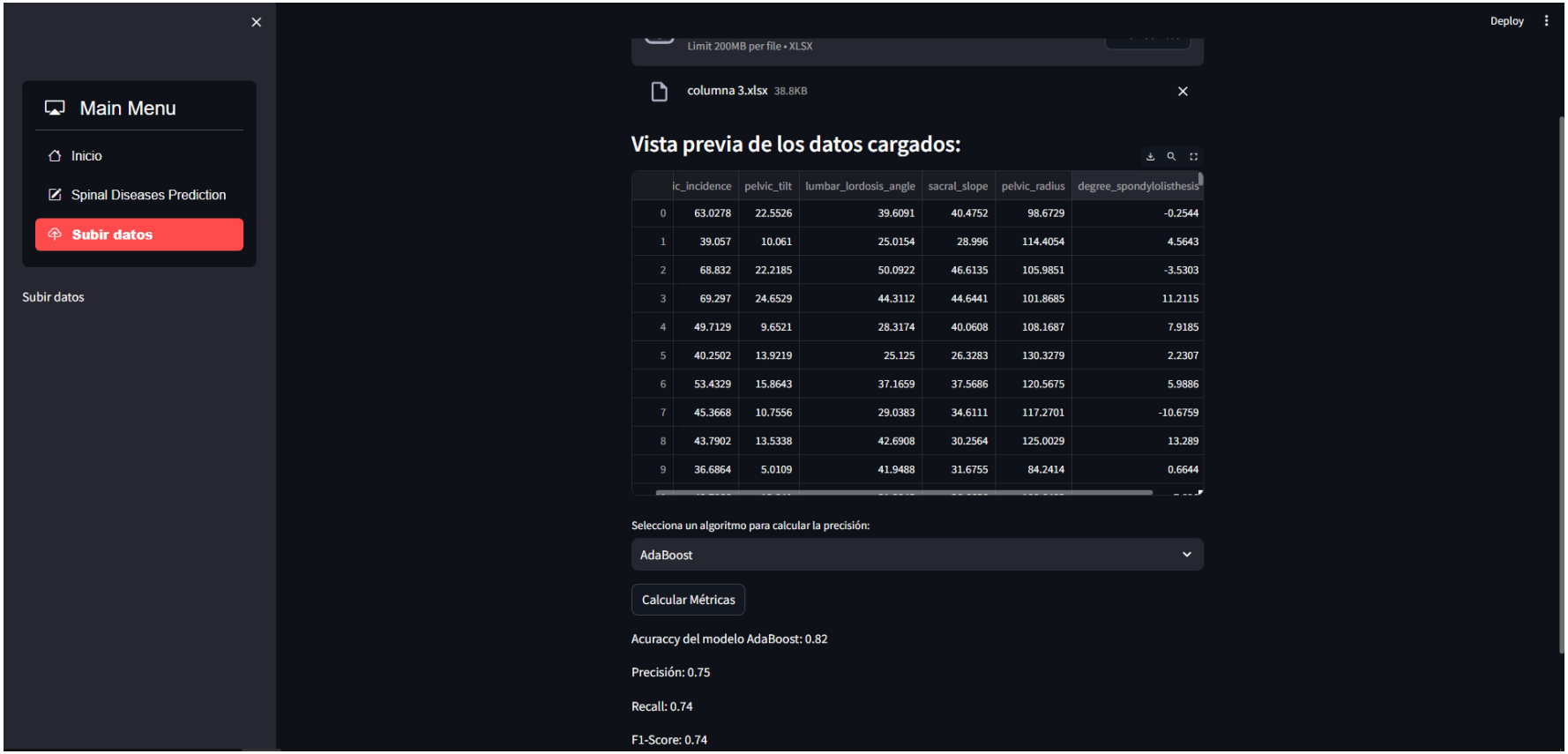
Figura N° 55: Resultado del Sistema Inteligente con el modelo Stacking 1



Fuente: Elaboración propia

Se visualiza que los datos que fueron llenados pertenecen a una persona con espondilolistesis. También se observa que se obtuvo un 0.85 de exactitud usando el modelo stacking 1.

Figura Nº 56: Resultados mediante el uso de la opción Subir Datos



Fuente: Elaboración propia

Se visualiza la evaluación de las métricas en función de toda la data subida al sistema inteligente, en donde obtuvo una exactitud de 0.82, una precisión 0.75, un recall 0.74 y una medida f1-score de 0.74.

V. DISCUSION

A continuación, se presentan las discusiones basadas en los resultados obtenidos durante el desarrollo del estudio en base a las evidencias recopiladas durante el estudio, así como la comparación e interpretación detallada de los hallazgos y su relevancia en el contexto de la investigación.

En cuanto al objetivo general, el cual es aplicar un sistema inteligente con ensemble machine learning con el método stacking mediante el uso de balanceo de datos que permitirá predecir enfermedades de la columna vertebral (Hernia y espondilolistesis), se aplicó la metodología KDD el cual permitirá la extracción de patrones útiles, información y conocimientos previamente desconocidos desde el conjunto de datos. Mediante este método se simplificó el proceso y la construcción de los modelos de machine learning. En base a los antecedentes, ningún autor utilizo la metodología KDD, por lo cual en la investigación que se está realizando genera un enfoque distinto y mucho más ordenado en la aplicación de machine learning.

Para desarrollar del modelo predictivo se utilizó la base de datos SQL, para el almacenamiento y manipulación de datos, para el análisis estadístico, Jupyter como plataforma que se basa en web interactiva compatible con el lenguaje de programación Python y como entorno de desarrollo Spyder.

En lo que respecta al objetivo específico 1, a partir de la comparación entre los algoritmos de aprendizaje automático con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA. se observó que los mejores modelos se obtuvieron utilizando los algoritmos AdaBoost (91.11%), XGBoost (88.71%) y por otro lado, mediante el uso de los modelos stacking, los modelos stacking 1, stacking 3, stacking 4 y stacking 5 tuvieron un 88.71% de exactitud. En la investigación realizada por Martinez et.al, en donde utilizo la métrica de Acuraccy para poder evaluar el rendimiento de su modelo basado en redes neuronales para la detección de escoliosis en imagen rayos x de columna, en donde agrego una clase de pacientes sanos para obtener un mejor Acuraccy (94%). En base a ello se tomó en cuenta las personas que no poseen ninguna enfermedad de la columna vertebral para la aplicación de ML en el sistema inteligente y también se resalta que fue el único autor que uso una sola métrica de evaluación que es el Acuraccy.

En lo que respecta al objetivo específico 2, a partir de la comparación entre los algoritmos de aprendizaje automático con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA se observó que los mejores modelos se obtuvieron utilizando los algoritmos AdaBoost (90.29%) y stacking 1, stacking 3, stacking 4, stacking 5, los cuales destacaron en la aplicación de la métrica de Precisión, obteniendo los resultados de 86.40% y 84.19% todos los modelos stacking mencionados. Con lo mencionado anteriormente, se obtuvieron resultados de investigaciones de zhang et al. (2023) en donde uso los algoritmos XGB, RL, RF, ADA, KNN, SVM, NB y multilayer perceptrón, pero el mejor algoritmo fue XGB y obtuvo un 85.71%. Asimismo, Varcin (2021) uso el algoritmo RNC, donde obtuvo un 99%. De la misma manera Ansari (2013) que utilizo la misma data de esta investigación y uso 2 tipos de redes neuronales y SVM y obtuvo un 93.87. Asimismo, Karabacak, Margetis, usaron los algoritmos XGB, LightGBM, RF y CatBoost, donde obtuvieron un 87.8%. Finalmente, Unal Y., Polat K. y Erdinc H. (2014) también uso la misma data y usaron los algoritmos de NB, KNN, SVM y multilayer perceptrón, donde tuvieron los resultados de 90%, 80%, 83% y 85%.

Al contrario, en la investigación de los autores Martinez et al. (2022), solo utilizaron la métrica de Acuraccy, en la investigación de Castro et al. (2020) y Chen et al. (2018) no consideraron la métrica de precisión, ya que usaron las métricas de validación cruzada, recall y AUC ROC.

En lo que respecta al objetivo específico 3, a partir de la comparación entre los algoritmos de aprendizaje automático con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA se observó que los mejores modelos se obtuvieron utilizando los algoritmos AdaBoost (90.35%) y Regresión Logística (87.03%). Por otro lado, Random Forest, XGboost, stacking 1, stacking 3, stacking 4 y stacking 5 destacaron en la aplicación de la métrica de Recall (Sensibilidad), obteniendo el mismo resultado de 85.88% de sensibilidad. Los autores Varcin y Chen obtuvieron rendimientos diferentes en su empleo de la métrica recall, para la evaluación de su modelo. En la investigación de Varcin obtuvo un 98% de recall, pero por otro lado en la investigación de Chen solo obtuvo un 69.84% de recall. Asimismo, Castro et al (2020) uso los algoritmos KNN y RF donde obtuvo un 92.9% y 80.6% respectivamente. Finalmente, Unal Y., Polat K. y Erdinc H. (2014) usaron

los algoritmos multilayer perceptrón, KNN, NB y SVM donde obtuvieron un respectivamente.

Al contrario, en las investigaciones de los autores Martinez, Zhang et al. (2023), Ansari (2013) y Karabacak, y Margetis (2023), no consideraron esta métrica en sus investigaciones, ya que se enfocaron en medidas similares como Acuraccy, intervalo de confianza, auc roc y precisión.

En lo que respecta al objetivo específico 4, a partir de la comparación entre los algoritmos de aprendizaje automático con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA se observó que los mejores modelos se obtuvieron utilizando los algoritmos AdaBoost y Regresión Logística destacaron en la aplicación de la métrica de F1-score, obteniendo un 89.97% y 86.37% de score. Por otro lado, los modelos stacking 1, stacking 3, stacking 4 y stacking 5 obtuvieron un 84.83% de score, haciendo referencia a otros autores que usaron como base para sus investigaciones. Mediante lo mencionado anteriormente, los autores Unal Y., Polat K. y Erdinc H. (2014) usaron los algoritmos multilayer perceptrón (0.7738), KNN (0.7021), Naive bayes (0.7263 y svm (0.7298). Finalmente usaron los algoritmos multilayer perceptrón, KNN, Naive bayes y SVM, donde obtuvieron los resultados de 90%, 88%, 95% y 90% respectivamente.

En lo que respecta al objetivo específico 5, a partir de la comparación entre los algoritmos de aprendizaje automático con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA se observó que el algoritmo que tuvo mejor resultado de AUC es AdaBoost (0.98) y los modelos Random Forest, Regresión logística, SVM y XGBoost obtuvieron un 0.96 de AUC. Realizando una comparación entre los modelos stacking, el modelo stacking 5 obtuvo mejores resultados frente a la métrica de AUC, teniendo resultados cercanos a los mejores modelos propuestos. La métrica de AUC es una de las métricas más usadas por los investigadores, como los autores Zhang y Chen que obtuvieron un 87% y 0,7495 a 0,7590 (74.95% a 75.90%). Asimismo, Castro (2020) uso los algoritmos KNN y RF donde se obtuvo un 0.97 y 0.91. Por otro lado, Karabacak, Margetis () usaron los algoritmos XGBoost, LightGBM, CatBoost y Random Forest donde obtuvieron un 0.814 y finalmente los autores Unal Y., Polat K. y Erdinc H. (2014) donde uso

multilayer perceptrón, KNN, NB y RF donde obtuvieron los resultados de 0.989, 0.946, 0.999 y 0.953.

Al contrario, en las investigaciones de Martínez (2022), Varcin (2021) y Ansari (2013) no consideraron esta métrica en sus investigaciones, ya que se enfocaron en medidas similares como Acuraccy (exactitud), precisión, recall y especificidad

En lo que respecta al objetivo específico 6, a partir de la comparación entre los algoritmos de aprendizaje automático con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA se observó que los mejores modelos se obtuvieron utilizando AdaBoost (95.60%) y los modelos XGBoost y Random Forest obtuvieron el mismo porcentaje de especificidad de 95.14%., superando así a los modelos stacking 1, stacking 3, stacking 4 y stacking 5 que obtuvieron un 95.01% de especificidad. Con lo mencionado anteriormente, en similitud con las investigaciones, Varcin (2021) uso el algoritmo de redes neuronales convolucionales donde obtuvo una especificidad de 98% y por otro lado, los autores Unal Y., Polat K. y Erdinc H. (2014) usaron los algoritmos multilayer perceptrón, KNN, Naive bayes y SVM, donde obtuvieron los resultados de 93.69%, 90.9%, 95.50% y 91.89% respectivamente en función a la especificidad.

Al contrario, los autores Martinez (2022), Zhang et al. (2023), Ansari (2013), Castro et al (2020), Chen et al. (2018) y Karabacak, Margetis (2023), no consideraron esta métrica, ya que usaron otras métricas como AUC ROC, precisión, recall, f1-score, intervalo de confianza y Acuraccy.

En lo que respecta al objetivo específico 7, a partir de la comparación entre los algoritmos de aprendizaje automático con 12 algoritmos DT, GB, KNN, NB, NC, RF, RN, RNC, RL, SVM, XGB y ADA se observó que los algoritmos que tuvieron un mejor porcentaje en función a la métrica MCC es AdaBoost (87.37%). Por otro lado, los modelos XGBoost y Random Forest obtuvieron el mismo porcentaje de 85.18%. Realizando una comparación de los modelos stacking y los algoritmos planteados para crear los modelos stacking, los algoritmos AdaBoost, XGBoost y Random Forest superaron a los modelos stacking propuestos. Se puede apreciar que ninguna investigación realizada por otro autor incluyo la métrica MCC para

evaluar el rendimiento de su modelo. En base a ello, se da a conocer esta métrica de evaluación, para generar un aporte en la evaluación de los modelos.

Al contrario, los autores Martinez (2022), Zhang et al. (2023), Ansari (2013), Castro et al (2020), Chen et al. (2018), Unal Y., Polat K. y Erdinc H. (2014), Varcin (2021) y Karabacak & Margetis (2023), no consideraron esta métrica, ya que usaron otras métricas como AUC ROC, precisión, especificidad, recall (especificidad), f1-score, intervalo de confianza y Acuraccy.

En lo que respecta al objetivo específico 8, se aplicó específicamente a los algoritmos de Redes neuronales y redes neuronales convolucionales. En donde se obtuvo una pérdida de 2.30 y 2.60 correlativamente a los algoritmos mencionados. También se aplicó la fórmula de entropía categórica cruzada en donde se obtuvo una pérdida de 0.60 y 1.288. En las investigaciones de los autores Zhang, Tavana, Martinez, ninguno utilizó la función de loss (pérdida) en sus investigaciones, para poder evaluar la perdida que tuvieron sus algoritmos de redes neuronales.

VI. CONCLUSIONES

A partir de los resultados obtenidos en la investigación, el investigador presenta las siguientes conclusiones:

Primera: El estudio se centró en el desarrollo de un sistema inteligente basado en machine learning para predecir enfermedades de la columna vertebral (hernia, espondilolistesis). La metodología KDD fue aplicada, proporcionando un marco estructurado para el proceso de descubrimiento de conocimientos, en donde ayudó a optimizar el uso de recursos al enfocarse en la extracción de conocimiento y así permitiendo simplificar el proceso mediante modelos de aprendizaje automático. Autores previos respaldaron la eficacia de esta metodología, destacando la importancia de este enfoque para la extracción de conocimiento.

Segunda: La comparación de resultados entre los algoritmos permitió conocer que los modelos que obtuvieron un mejor Acuraccy son AdaBoost (91.11%), XGBoost (88.71%) y por otro lado, mediante el uso de los modelos stacking, los modelos stacking 1, stacking 3, stacking 4 y stacking 5 tuvieron un 88.71% de exactitud. Realizando una comparación, los modelos stacking obtuvieron la misma exactitud que los modelos XGBoost y Random Forest. Estos resultados superaron a otros algoritmos como AdaBoost, Regresión Logística, Decision tree, KNN, Nearest Centroid, Redes neuronales, redes neuronales convulsiónales y Gradient Boosting. En cuanto a la variabilidad de resultados de diferentes estudios, ciertos autores no incluyeron esta métrica en sus investigaciones.

Tercera: La comparación de resultados entre los algoritmos permitió conocer que los modelos basados en AdaBoost (90.29%) y stacking 1, stacking 3, stacking 4, stacking 5, los cuales destacaron en la aplicación de la métrica de Precisión, obteniendo los resultados de 86.40% y 84.19% todos los modelos stacking mencionados. Estos resultados superaron a otros algoritmos como XGBoost, SVM, AdaBoost, Regresión Logística, Decision tree, KNN, Nearest Centroid, Redes neuronales, redes neuronales convulsiónales y Gradient Boosting. En cuanto a la variabilidad de resultados de diferentes estudios, ciertos autores no incluyeron esta métrica en sus investigaciones.

Cuarta: La comparación de resultados entre los algoritmos permitió conocer que los modelos basados en AdaBoost (90.35%) y Regresión Logística (87.03%). Por otro lado, Random Forest, XGBoost, stacking 1, stacking 3, stacking 4 y stacking 5 destacaron en la aplicación de la métrica de Recall (Sensibilidad), obteniendo el mismo resultado de 85.88% de sensibilidad. Estos resultados superaron a otros algoritmos como SVM, AdaBoost, Naive Bayes, Regresión Logística, Decision tree, KNN, Nearest Centroid, Redes neuronales, redes neuronales convulsiónales y Gradient Boosting. En cuanto a la variabilidad de resultados de diferentes estudios, ciertos autores no incluyeron esta métrica en sus investigaciones.

Quinta: La comparación de resultados entre los algoritmos permitió conocer que el modelo de AdaBoost y Regresión Logística destacaron en la aplicación de la métrica de F1-score, obteniendo un 89.97% y 86.37% de score. Por otro lado, los modelos stacking 1, stacking 3, stacking 4 y stacking 5 obtuvieron un 84.83% de score. Estos resultados superaron a otros algoritmos como SVM, Naive Bayes, XGBoost, AdaBoost, Regresión Logística, Decision tree, KNN, Nearest Centroid, Redes neuronales, redes neuronales convulsiónales y Gradient Boosting. En cuanto a la variabilidad de resultados de diferentes estudios, ciertos autores no incluyeron esta métrica en sus investigaciones.

Sexta: La comparación de resultados entre los algoritmos permitió conocer que los modelos basados en AdaBoost (0.98) y los modelos Random Forest, Regresión logística, SVM y XGBoost obtuvieron un 0.96 de AUC. Realizando una comparación entre los modelos stacking, el modelo stacking 5 obtuvo mejores resultados frente a la métrica de AUC, teniendo resultados cercanos a los mejores modelos propuestos. Estos resultados superaron a otros algoritmos como SVM, AdaBoost, Regresión Logística, Decision tree, KNN, Nearest Centroid, Redes neuronales, redes neuronales convolucionales y Gradient Boosting. En cuanto a la variabilidad de resultados de diferentes estudios, ciertos autores no incluyeron esta métrica en sus investigaciones.

Septima: La comparación de resultados entre los algoritmos, dio como resultados que los algoritmos que tuvieron una mejor especificidad son AdaBoost (95.60%) y los modelos XGBoost y Random Forest obtuvieron el mismo porcentaje de especificidad de 95.14%., superando así a los modelos stacking 1, stacking 3, stacking 4 y stacking 5 que obtuvieron un 95.01% de especificidad. Estos resultados superaron a otros algoritmos como Naive Bayes, XGBoost, AdaBoost, Regresión Logística, Decision tree, KNN, Nearest Centroid, Redes neuronales, redes neuronales convulsiónales y Gradient Boosting. En cuanto a la variabilidad de resultados de diferentes estudios, ciertos autores no incluyeron esta métrica en sus investigaciones.

Octava: La comparación de resultados entre los algoritmos, permitió conocer que el algoritmo que tuvo un mejor porcentaje en función a la métrica MCC (Matthews Correlation Coefficient) es AdaBoost (87.37%). Estos resultados superaron a otros algoritmos como Naive Bayes, SVM, AdaBoost, Regresión Logística, Decision tree, KNN, Nearest Centroid, Redes neuronales, redes neuronales convulsiónales y Gradient Boosting. En cuanto a la variabilidad de resultados de diferentes estudios, ciertos autores no incluyeron esta métrica en sus investigaciones.

Novena: En función a la métrica de Loss (Pérdida), se aplicó a los modelos de redes neuronales y redes neuronales convulsiónales, ya que sirvió como criterio para ajustar los parámetros del modelo. Los modelos obtuvieron los siguientes resultados de 3.24 y 2.60 de pérdida correlativamente. Para la aplicación de la función de la pérdida se eligió la fórmula de entropía cruzada categórica en donde los modelos obtuvieron 0.285 y 1.288 de pérdida. Realizando una comparación se concluye que el modelo de redes neuronales tuvo menos pérdida que el modelo de redes neuronales convolucionales.

Décima: Mediante la comparación de los modelos stacking ensemble, se obtuvo que los modelos que tuvieron mejor rendimiento en función de todas las métricas son los Stacking 1, Stacking 3 y Stacking 5

VII. RECOMENDACIONES

Después de haber generado las conclusiones, se procedió a formular las recomendaciones detallando específicamente las acciones sugeridas para mejorar y optimizar los aspectos identificados durante el estudio para futuras investigaciones:

Primera: Se recomienda explorar diferentes metodologías además de KDD, como CRISP-DM, SEMMA o TDSP ya que permitirá tener una visión más completa de los distintos enfoques que poseen otras metodologías.

Segunda: Se recomienda mejorar el proceso de evaluación de enfermedades de la columna entre el sistema inteligente y la base de datos, ya que, al momento de generar nuevos resultados el sistema inteligente no agrega los nuevos datos registrados a la base de datos, en base a ello se recomienda automatizar el sistema inteligente y la base de datos para que se pueda agregar los nuevos datos registrados a la base de datos. De esta forma, el modelo entrenaría de manera automática y generaría resultados más precisos.

Tercera: Se recomienda aplicar y comparar con otros modelos como Support Vector Regression (SVR), Redes Neuronales de Atención (Attention Networks), Redes Generativas Adversariales (GANs), Redes Neuronales Evolutivas (Neuroevolution) o incluso usar herramientas o plataformas de AutoML como Google AutoML, H2O.ai, Auto-sklearn, y TPOT, entre otras. Las herramientas mencionadas previamente ofrecen interfaces de usuario amigables y ayudan a los usuarios a través del proceso de desarrollo del modelo sin requerir una experiencia en aprendizaje automático. De esta manera se podría facilitar la aplicación de otros modelos.

Cuarta: Se recomienda comparar con otros estudios previamente realizados, en donde también tengan un enfoque de clasificación de enfermedades de la columna vertebral, lo cual contribuirá a mejorar la precisión de los modelos propuestos.

REFERENCIAS

Alarcón, C. (2015). Optimización del clasificador “Naive Bayes” usando árbol de decisión C4.5. Disponible en: https://cybertesis.unmsm.edu.pe/bitstream/handle/20.500.12672/4183/Alarc%C3%B3n_jc.pdf?sequence=3&isAllowed=y

Ansari, S., Sajjad, F., Naveed, N., Zia-ul-Qayyum, Shafi, I. & Ahmad, J. (2013). Diagnosis of vertebral column disorders using machine learning classifiers. Disponible en: https://www.researchgate.net/publication/261271432_Diagnosis_of_Vertebral_Column_Disorders_Using_Machine_Learning_Classifiers?enrichId=rgreq-fbaca00d3ef357b3a9fa8c39161c4f54XXX&enrichSource=Y292ZXJQYWdlOzI2MTI3MTQzMjtBUzoyMTA0MzI0NTIyMzkzNjNAMTQyNzE4MjQxNDUwNw%3D%3D&el=1_x_2&_esc=publicationCoverPdf

Bach, M. (2022). New Undersampling Method Based on the kNN Approach. *Procedia Computer Science*, *207*, 3397–3406. Disponible en: https://doi.org/10.1016/j.procs.2022.09.399

Bachmann, K. R. (2021). Spinal Deformities in the Adolescent Athlete. In *Clinics in Sports Medicine* (Vol. 40, Issue 3, pp. 541–554). W.B. Saunders. Disponible en: https://doi.org/10.1016/j.csm.2021.03.007

Bae, S. Y., Lee, J., Jeong, J., Lim, C., & Choi, J. (2021). Effective data-balancing methods for class-imbalanced genotoxicity datasets using machine learning algorithms and molecular fingerprints. *Computational Toxicology*, *20*. Disponible en: https://doi.org/10.1016/j.comtox.2021.100178

Bai, G. & Chandra, R. (2023). Gradient boosting Bayesian neural networks via Langevin MCMC. Disponible en: https://doi.org/10.1016/j.neucom.2023.126726

Birfir, S., Elalouf, A. & Rosenbloom, T. (2023). Building machine-learning models for reducing the severity of bicyclist road traffic injuries. Disponible en: https://doi.org/10.1016/j.treng.2023.100179

Bonilla, C. (2020). REDES CONVOLUCIONALES. Disponible en: https://idus.us.es/bitstream/handle/11441/115221/TFG%20DGMyE%20Bonilla%20Carri%C3%B3n%2C%20Carmelo.pdf?sequence=1&isAllowed=y

Borja-Robalino, R., Monleon-Getino, A., Monleón-Getino, A., & Rodellar, J. (2020). *Estandarización de Métricas de Rendimiento para Clasificadores Machine y Deep Learning Predective Models in Epidemiology View project MetaCompMicro View project Estandarización de métricas de rendimiento para clasificadores Machine y Deep Learning*. Disponible en: https://www.researchgate.net/publication/339943922

Bravo, S. & Cruz, J. (2015). Estudios de exactitud diagnóstica: Herramientas para su Interpretación. Disponible en: https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0717-93082015000400007&lng=en&nrm=iso&tlng=en

Cano, C., & Ruiz, E. (2018). *Clasificación: Aspectos Prácticos Medidas para evaluar y comparar el rendimiento de los clasificadores*. Disponible en: https://archive.ics.uci.edu/ml/datasets/breast+ca

Castro, R., Hae, E., Choi, Y., Yong, G. & Ko, S. (2020). Early detection of ankylosing spondylitis using texture features and statistical machine learning, and deep learning, with some patient age analysis. Disponible en: https://www.sciencedirect.com/science/article/pii/S0895611120300215

Castro Maldonado, J. J., Gómez Macho, L. K., & Camargo Casallas, E. (2023). La investigación aplicada y el desarrollo experimental en el fortalecimiento de las competencias de la sociedad del siglo XXI. *Tecnura*, *27*(75), 140–174. Disponible en: https://doi.org/10.14483/22487638.19171

Chamat, C. (2021). Modelo Predictivo de Deserción Estudiantil de Educación Preescolar, Básica y Media en el Municipio de Medellin. Disponible en: https://bibliotecadigital.udea.edu.co/bitstream/10495/25045/10/ChamatCelger_2021_ModeloPredictivoEducativo.pdf

Charles, Y. P., Lamas, V., & Ntilikina, Y. (2023). Artificial intelligence and treatment algorithms in spine surgery. In *Orthopaedics and Traumatology: Surgery and Research* (Vol. 109, Issue 1). Elsevier Masson s.r.l. Disponible en: https://doi.org/10.1016/j.otsr.2022.103456

Chen, T. & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Disponible en: http://dx.doi.org/10.1145/2939672.2939785

Chen, Y. F., Lin, C. S., Wang, K. A., Rahman, L. O. A., Lee, D. J., Chung, W. S., & Lin, H. H. (2018). Design of a clinical decision support system for fracture prediction using imbalanced dataset. *Journal of Healthcare Engineering*, *2018*. Disponible en: https://doi.org/10.1155/2018/9621640

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*(1). Disponible en: https://doi.org/10.1186/s12864-019-6413-7

Chicco, D., Starovoitov, V., & Jurman, G. (2021). The Benefits of the Matthews Correlation Coefficient (MCC) over the Diagnostic Odds Ratio (DOR) in Binary Classification Assessment. *IEEE Access*, *9*, 47112–47124. Disponible en: https://doi.org/10.1109/ACCESS.2021.3068614

Chicco, D., Tötsch, N., & Jurman, G. (2021). The matthews correlation coefficient (Mcc) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Mining*, *14*, 1–22. Disponible en: https://doi.org/10.1186/s13040-021-00244-z

Ciampiconi, L., Elwood, A., Leonardi, M., Mohamed, A., & Rozza, A. (2023). *A survey and taxonomy of loss functions in machine learning*. Disponible en: http://arxiv.org/abs/2301.05579

Cruz-Ruiz, F. de la, Canul-Reich, J., Rivera-López, R., & Cruz-Hernández, E. de la. (2023). Impact of data balancing a multiclass dataset before the creation of association rules to study bacterial vaginosis. *Intelligent Medicine*. Disponible en: https://doi.org/10.1016/j.imed.2023.02.001

Cui Y, Zhu J, Duan Z, Liao Z, Wang S, Liu W. Artificial Intelligence in Spinal Imaging: Current Status and Future Directions. *International Journal of Environmental Research and Public Health*. 2022; 19(18):11708. https://doi.org/10.3390/ijerph191811708

Dalianis, H. (2018). Evaluation Metrics and Evaluation. In *Clinical Text Mining* (pp. 45–53). Springer International Publishing. Disponible en: https://doi.org/10.1007/978-3-319-78503-5_6

Dou, J., Yunus, A. P., Bui, D. T., Merghadi, A., Sahana, M., Zhu, Z., Chen, C. W., Han, Z., & Pham, B. T. (2020). Improved landslide assessment using support vector machine with bagging, boosting, and stacking ensemble machine learning framework in a mountainous watershed, Japan. *Landslides*, *17*(3), 641–658. Disponible en: https://doi.org/10.1007/s10346-019-01286-5

Dymaxion Labs. (2021). Capacitación Data4Now. Disponible en: https://unstats.un.org/capacity-development/data-for-now/training-materials/Aprendizaje-Automatico.pdf

Freund, Y & Schapire, R. (1996). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Disponible en: https://www.sciencedirect.com/science/article/pii/S002200009791504X

Gallucci, M., Limbucci, N., Paonessa, A., & Splendiani, A. (2007). Degenerative Disease of the Spine. In *Neuroimaging Clinics of North America* (Vol. 17, Issue 1, pp. 87–103). Disponible en: https://doi.org/10.1016/j.nic.2007.01.002

García, J. (2021). Comparativa de técnicas de balanceo de datos. Aplicación a un caso real para la predicción de fuga de clientes. Disponible en: https://digibuo.uniovi.es/dspace/bitstream/handle/10651/60629/TFM_Joaqu%C3%ADnGarc%C3%ADaAbad.pdf?sequence=4

Gómez, A. & Gómez, K. (2019). Muestreo estadístico para docentes y estudiantes. Disponible en: https://tauniversity.org/sites/default/files/ebook_muestreo_estadistico_para_docentes_y_estudiantes._dr._angel_gomez_degraves_y_prof._karine_gomez_marquina.pdf

Gómez, C. (2021). HALLAZGOS RADIOLÓGICOS EN LA COLUMNA LUMBAR MEDIANTE RAYOS X DIGITAL SAN BORJA, 2018. Disponible en: http://repositorio.unfv.edu.pe/bitstream/handle/20.500.13084/4987/UNFV_G%C3%93MEZ_DAVILA_CESAR_AUGUSTO_TITULO_LICENCIADO_2021.pdf?sequence=1&isAllowed=y

González, J., Rodríguez, J., De la Puente, E. & Díaz, M. (2000). TRATAMIENTO DE LA COLUMNA VERTEBRAL EN LA EDUCACION SECUNDARIA OBLIGATORIA: PARTE I – PREVENCION Y EJERCICIOS POCO RECOMENDABLES. Disponible en: https://repositorio.uam.es/bitstream/handle/10486/3779/26088_3.pdf?sequence=1

González, R., Barrientos, A., Toapanta, M., & Del Cerro, J. (2017). Aplicación de las Máquinas de Soporte Vectorial (SVM) al diagnóstico clínico de la Enfermedad de Párkinson y el Temblor Esencial. *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, *14*(4), 394–405. Disponible en: https://doi.org/10.1016/j.riai.2017.07.005

Grabner, C. (2020). Getting started with the Spyder IDE. Disponible en: https://claudius-graebner.com/files/abm-python-course-dt/intro-spyder.pdf

Gualdrón, O. (2006). Desarrollo de diferentes métodos de selección de variables para sistemas multisensoriales. Disponible en: https://www.tdx.cat/bitstream/handle/10803/8473/Tesis_Oscar_Gualdron.pdf?sequence=1

Hayati, R., Arip, A., Lukitaningsih, E., Earlia, N., Karma, T. & Idroes, R. (2023). Combination of PCA with LDA and SVM classifiers: A model for determining the geographical origin of coconut in the coastal plantation, Aceh Province, Indonesia. Disponible en: https://doi.org/10.1016/j.cscee.2023.100552

Hari, A., Abdurrahman, F & Afif, A. (2020). Nearest Centroid Classifier with Centroid-Based Outlier Removal for Classification. Disponible en: https://www.researchgate.net/publication/339576432_Nearest_Centroid_Classifier_with_Outlier_Removal_for_Classification

He, Y., Cheng, Y., Ma, M., Li, F., Song, Y., Liu, L., Wang, X. & Huang, J. (2022). A Novel Design Concept of Cemented Paste Backfill (CPB) Materials: Biobjective Optimization Approach by Applying an Evolved Random Forest Model. Disponible en: https://doi.org/10.3390/ma15238298

Infanzon, R. & Riveros, J. (2022). ALTERACIONES POSTURALES DE LA COLUMNA Y CEFALEA EN ESTUDIANTES DE LA UNIVERSIDAD PERUANA LOS ANDES – HUANCAYO 2021. Disponible en: https://repositorio.upla.edu.pe/handle/20.500.12848/5058

Irene Moral Peláez. (n.d.). Modelos de regresión: lineal simple y regresión logística. Disponible en: https://revistaseden.org/files/14-CAP%2014.pdf

Jiménez, S. & Merino, A. (2022). CRISP-DM- Based Machine Learning Models For Analyzing the Depression Level in Students of the National Polythecnic School. Disponible en: https://lajc.epn.edu.ec/index.php/LAJC/article/view/335

Kandelwal, Y. (2021). Ensemble Stacking for Machine Learning and Deep Learning. Disponible en:

https://www.analyticsvidhya.com/blog/2021/08/ensemble-stacking-for-machine-learning-and-deep-learning/

Karabacak, M. & Margetis, K. (2023). Machine Learning-Based Prediction of Short-Term Adverse Postoperative Outcomes in Cervical Disc Arthroplasty Patients. Disponible en: https://doi.org/10.1016/j.wneu.2023.06.025

Kim Soon, G., Kim On, C., Mohd Rusli, N., Soo Fun, T., Alfred, R., & Tse Guan, T. (2020). Comparison of simple feedforward neural network, recurrent neural network and ensemble neural networks in phishing detection. *Journal of Physics: Conference Series*, *1502*(1). Disponible en: https://doi.org/10.1088/1742-6596/1502/1/012033

Leevy, J. L., Johnson, J. M., Hancock, J., & Khoshgoftaar, T. M. (2023). Threshold optimization and random undersampling for imbalanced credit card data. *Journal of Big Data*, *10*(1). Disponible en: https://doi.org/10.1186/s40537-023-00738-z

Li, A., Liu, M., & Sheather, S. (2023). Predicting stock splits using ensemble machine learning and SMOTE oversampling. *Pacific Basin Finance Journal*, *78*. Disponible en: https://doi.org/10.1016/j.pacfin.2023.101948

Liang, R., Yip, J., Fan, Y., Cheung, J. P. Y., & To, K. T. M. (2022). Electromyographic Analysis of Paraspinal Muscles of Scoliosis Patients Using Machine Learning Approaches. *International Journal of Environmental Research and Public Health*, *19*(3). Disponible en: https://doi.org/10.3390/ijerph19031177

Liu, Y., Cheng, J., Yan, C., Wu, X., & Chen, F. (2015). Research on the Matthews correlation coefficients metrics of personalized recommendation algorithm evaluation. *International Journal of Hybrid Information Technology*, *8*(1), 163–172. Disponible en: https://doi.org/10.14257/ijhit.2015.8.1.14

Lööv, S. (2020). COMPARISON OF UNDERSAMPLING METHODS FOR PREDICTION OF CASTING DEFECTS BASED ON PROCESS PARAMETERS. Disponible en: https://www.diva-portal.org/smash/get/diva2:1597599/FULLTEXT01.pdf

Ma, Z., Wang, P., Gao, Z., Wang, R., & Khalighi, K. (2018). Ensemble of machine learning algorithms using the stacked generalization approach to estimate the warfarin dose. *PLoS ONE*, *13*(10). Disponible en: https://doi.org/10.1371/journal.pone.0205872

Martínez, G. (2022). Desarrollo de un clasificador basado en redes neuronales para la detección de la escoliosis en imagen RX de columna. Disponible en: http://repositorio.uan.edu.co/bitstream/123456789/6085/3/2022_Trabajo.G.Martinez%2CGonzalo.pdf.

Martínez Pérez, J. A., & Pérez Martin, P. S. (2023). ROC curve. In *Semergen* (Vol. 49, Issue 1). Ediciones Doyma, S.L. Disponible en: https://doi.org/10.1016/j.semerg.2022.101821

Marzal, A., & Gracia, I. (2009). Introducción a la programación con Python. Disponible en: https://www.u-cursos.cl/ingenieria/2011/2/CC3501/1/material_docente/bajar?id_material=381752

McGrath, K., Lee, J., Steinmetz, M., Lonser, R. R., & Resnick, D. K. (2022). Degenerative Spine Disorders and Multiple Sclerosis. In *Neurologic Clinics* (Vol. 40, Issue 2, pp. 249–259). W.B. Saunders. Disponible en: https://doi.org/10.1016/j.ncl.2021.11.004

Menasalvas, E., Rodriguez, A., Guzman, M., Jimenez, S., Duque, S. (n.d.). Newsletter Trimestral-Algoritmos. Disponible en: Disponible en: https://blogs.upm.es/catedra-idanae/wp-content/uploads/sites/698/2020/01/Idanae-ESP-4T19-LR.pdf

Mendoza Olguín, G. E., Laureano de Jesús, Y., & Pérez de Celis Herrero, M. de la C. (2019). Métricas de similaridad y evaluación para sistemas de recomendación de filtrado colaborativo. *Revista de Investigación En Tecnologías de La Información*, *7*(14), 224–240. Disponible en: https://doi.org/10.36825/riti.07.14.019

Mohammed, R., Rawashdeh, J., & Abdullah, M. (2020). Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. *2020 11th International Conference on Information and Communication Systems, ICICS 2020*, 243–248. Disponible en: https://doi.org/10.1109/ICICS49469.2020.239556

Molina, M. (2020). degenerative lumbar spinal stenosis: basic concepts, clinical evaluation, treatment. *Revista Medica Clinica Las Condes*, *31*(5–6), 441–447. Disponible en: https://doi.org/10.1016/j.rmclc.2020.08.002

Naumetc, D. (n.d.). Artificial Neural Networks in plane control. Disponible en: https://api.core.ac.uk/oai/oai:www.theseus.fi:10024/123282

Nebot, A. C. (2018). Creating a predictive statistical model to determine mitigation functions in spoken Spanish. In *RILCE* (Vol. 34, Issue 3, pp. 1009–1027). Servicio de Publicaciones de la Universidad de Navarra. Disponible en: https://doi.org/10.15581/008.34.3.1009-27

Noriega-Álvarez, E., Domínguez Gadea, L., Sanz Viedma, S., del Prado Orduña Diez, M., Minoves Font, M., Peiró Valgañón, V., & García Jiménez, R. (2021). Nuclear Medicine in the diagnosis of pathologies of the spine: role of hybrid imaging. *Revista Espanola de Medicina Nuclear e Imagen Molecular*, *40*(1), 37–49. Disponible en: https://doi.org/10.1016/j.remn.2020.08.011

Nwagu, C., Kenneth, O., & Chinecherem, I. (n.d.). *Knowledge Discovery in Databases (KDD): An Overview*. Disponible en: https://sites.google.com/site/ijcsis/

Otzen, T., & Manterola, C. (2017). Técnicas de Muestreo sobre una Población a Estudio Sampling Techniques on a Population Study. In *Int. J. Morphol* (Vol. 35, Issue 1). Disponible en: http://dx.doi.org/10.4067/S0717-95022017000100037.

Pereira, R. M., Costa, Y. M. G., & Silla, C. N. (2020). MLTL: A multi-label approach for the Tomek Link undersampling algorithm: MLTL: The Multi-Label Tomek Link. *Neurocomputing*, *383*, 95–105. Disponible en: https://doi.org/10.1016/j.neucom.2019.11.076

Pineda, J. M. (2022). Predictive models in health based on machine learning. *Revista Medica Clinica Las Condes*, *33*(6), 583–590. Disponible en: https://doi.org/10.1016/j.rmclc.2022.11.002

Rajiv, D, et al. (2022). Dolor lumbar. Disponible en: https://www.clinicalkey.es/#!/content/book/3-s2.0-B978841382065100050X?scrollTo=%23hl0000867

Ramírez, C. (2020). APLICACIÓN DEL MACHINE LEARNING EN AGRICULTURA DE PRECISIÓN APPLICATION OF MACHINE LEARNING IN PRECISION AGRICULTURE. In *Revista Cintex* | (Vol. 25, Issue 2). Disponible en: https://revistas.pascualbravo.edu.co/index.php/cintex/article/view/356

Ramos, O. (2020). Métricas de Evaluación. Disponible en: http://oramosp.epizy.com/teaching/202/topicos/clases/4_MetricasEvaluacion.pdf

Ramos-Galarza, C. (2021). Editorial: Diseños de investigación experimental. *CienciAmérica*, *10*(1), 1–7. Disponible en: https://doi.org/10.33210/ca.v10i1.356

Randolph, G. B., Arya, ;, & Shamie, N. (n.d.). Anatomía de la columna vertebral. Disponible en: https://acreditacion-fmc.org/AAOS/pdf/Seccion_6.pdf

Rolon, D., et al. (2020). Introduction to Anaconda and Python: Installation and setup. Disponible en: https://doi.org/10.20982/tqmp.16.5.S003

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., & Church, K. (2020). Introduction to Anaconda and Python: Installation and setup. *The Quantitative Methods for Psychology*, *16*(5), S3–S11. Disponible en: https://doi.org/10.20982/tqmp.16.5.s003

Romei, A., Ruggieri, S., & Turini, F. (2006). KDDML: A middleware language and system for knowledge discovery in databases. *Data and Knowledge Engineering*, *57*(2), 179–220. Disponible en: https://doi.org/10.1016/j.datak.2005.04.007

Saghari, A., Budinská, I., Hosseinimehr, M., & Rahmani, S. (2023). A Robust-Reliable Decision-Making Methodology Based on a Combination of Stakeholders' Preferences Simulation and KDD Techniques for Selecting Automotive Platform Benchmark †. *Symmetry*, *15*(3). Disponible en: https://doi.org/10.3390/sym15030750

Sabab, Kabir, N., Biswas, A. Nazneen, T. & Shorif, M. (2021). An in-depth analysis of machine learning approaches to predict depression. Disponible en: https://doi.org/10.1016/j.crbeha.2021.100044

Satapathy, S. K., Bhoi, A. K., Loganathan, D., Khandelwal, B., & Barsocchi, P. (2021). Machine learning with ensemble stacking model for automated sleep staging using dual-channel EEG signal. *Biomedical Signal Processing and Control*, *69*. Disponible en: https://doi.org/10.1016/j.bspc.2021.102898

Sjeklocha, L. & Gatz, J. (2021). Traumatic Injuries to the Spinal Cord and Peripheral Nervous System. Disponible en: https://doi.org/10.1016/j.emc.2020.09.001

Sihab, Zahid, Rahman, S., Sarker, S., Ahmmad, E. Muyeen & Das, S. (2022). On the protection of power system: Transmission line fault analysis based on an optimal machine learning approach. Disponible en: https://doi.org/10.1016/j.egyr.2022.07.163

Sruthi, E. (2023). Understand Random Forest Algorithms with Examples (Updates 2023). Disponible en: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Tao, C., Tao, T., He, S., Bai, X. & Liu, Y. (2023). Wind turbine blade icing diagnosis using B-SMOTE-Bi-GRU and RFE combined with icing mechanism. Disponible en: https://doi.org/10.1016/j.renene.2023.119741

Tavana, P., Akraminia, M., Koochari, A., & Bagherifard, A. (2023). An efficient ensemble method for detecting spinal curvature type using deep transfer learning and soft voting classifier. *Expert Systems with Applications*, *213*. Disponible en: https://doi.org/10.1016/j.eswa.2022.119290

Toledo, E. (n.d.). Técnicas de investigación cuantitativas y cualitativas FAD UAEMex. Disponible en: https://core.ac.uk/download/pdf/80531608.pdf

Unal, Y., Polat, K & Erdinc, H. (2014). Pairwise FCM based feature weighting for improved classification of vertebral column disorders. Disponible en: http://dx.doi.org/10.1016/j.compbiomed.2013.12.004

Varcin, F., Erbay, H., Cetin, E., Cetin, I. & Kultur, T. (2021). End-to-end Computerized diagnosis of spondylolisthesis using only lumbar X-rays. Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7887126/

Van Der Linden, S., Brown, M., Gensler, L. S., Kenna, T., Maksymowych, W. P., & Taylor, W. J. (2022). Espondilitis anquilosante y otras formas de

espondiloartritis axial. Disponible en: https://www.clinicalkey.es/#!/content/book/3-s2.0-B9788413820651000808

Villegas Cubas, J., & Niño, G. M. (n.d.). Modelo de machine learning en la detección de sitios web phishing. Disponible en: https://www.proquest.com/docview/2758392861/35872E02BD2B4B3APQ/1?accountid=37408

Wang, Q., Ma, Y., Zhao, K., & Tian, Y. (2022). A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science*, *9*(2), 187–212. Disponible en: https://doi.org/10.1007/s40745-020-00253-5

Xu, S., Sang, B., Zeng, L., & Zhao, L. (2023). Two-Lane DNN Equalizer Using Balanced Random-Oversampling for W-Band PS-16QAM RoF Delivery over 4.6 km. *Sensors*, *23*(10). Disponible en: https://doi.org/10.3390/s23104618

Yin, W., Kirkulak-Uludag, B., Zhu, D., & Zhou, Z. (2023). Stacking ensemble method for personal credit risk assessment in Peer-to-Peer lending. *Applied Soft Computing*, *142*. Disponible en: https://doi.org/10.1016/j.asoc.2023.110302

Yuan, L., Yu, S., Yang, Z., Duan, M., & Li, K. (2023). A data balancing approach based on generative adversarial network. *Future Generation Computer Systems*, *141*, 768–776. Disponible en: https://doi.org/10.1016/j.future.2022.12.024

Zhang, Y., Wan, D. H., Chen, M., Li, Y. L., Ying, H., Yao, G. L., Liu, Z. L., & Zhang, G. M. (2023). Automated machine learning-based model for the prediction of delirium in patients after surgery for degenerative spinal disease. *CNS Neuroscience and Therapeutics*, *29*(1), 282–295. Disponible en: https://doi.org/10.1111/cns.14002

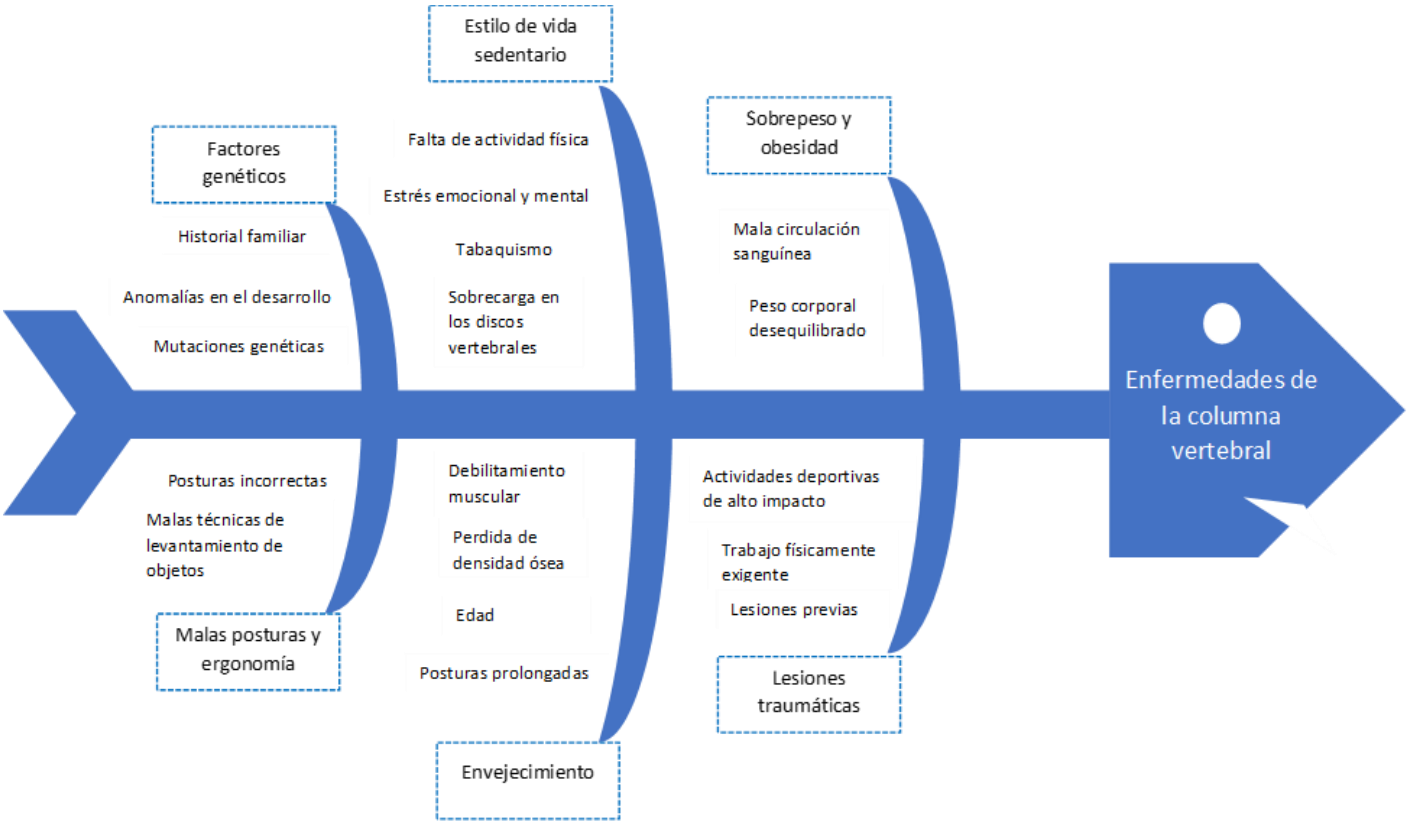
Zheng, Z., Cai, Y., Li, Y., Zheng, Z., Cai, Y., & Li, Y. (2015). OVERSAMPLING METHOD FOR IMBALANCED CLASSIFICATION. In *Computing and Informatics* (Vol. 34). Disponible en: https://www.cai.sk/ojs/index.php/cai/article/download/1277/724/8137

Zhu, W., He, X., Cheng, K., Zhang, L., Chen, D., Wang, X., Qiu, G., Cao, X., & Weng, X. (2019). Ankylosing spondylitis: etiology, pathogenesis, and treatments. In *Bone Research* (Vol. 7, Issue 1). Sichuan University. Disponible en: https://doi.org/10.1038/s41413-019-0057-8

ANEXOS

Anexo 1: Diagrama de causa y efecto

Figura Nº 57: Diagrama de Ishikawa



Fuente: Elaboración propia

Anexo 2: Matriz de Operacionalización de Variables

Tabla Nº 143: Matriz de operacionalización de variables

Variable	Definición conceptual	Definición operacional	Dimensiones	Indicadores	Escalas de Medición
Independiente: Machine learning	Según (Pineda, 2022) afirma que, el machine learning son técnicas computacionales modernas que permiten llegar a mejores resultados, sino que cada vez existe un uso más frecuente en la práctica clínica con el procesamiento en tiempo real, de la gran cantidad de datos que se maneje, permitiendo predecir cada vez con más exactitud distintas situaciones de salud, facilitando una rápida intervención; de forma inmediata para el área de emergencias y de forma preventiva para distintas posibles enfermedades.				
Dependiente: Predicción de enfermedades de la columna vertebral	Según (Gallucci et al., 2007), la enfermedad vertebral degenerativa ha sido estudiados por bioarqueología, y tienen un vínculo con el estilo de vida y el nivel de actividad física de las personas. La enfermedad degenerativa de la columna vertebral es una definición que abarca distintos	Según (Satapathy et al., 2021) menciona que es un método de ensamblaje, que se encuentra en la arquitectura de dos capas, los modelos y datos de la primera capa se consideran datos y los modelos de la capa base. La principal ventaja del	Métricas de evaluación	Exactitud (Acuraccy) $\frac{TP + TN}{TP + TN + FP´FN}$ (Aqil et al., 2022) Precisión $\frac{TP}{TP + FP}$ (Ramos Ponce, 2020) Exhaustividad $\frac{TP}{TP + FN}$ (Menasalvas et al., n.d.) F1-score	

	espectros de anomalías degenerativas, afecta a las estructuras óseas y al disco intervertebral, aunque distintas patologías están relacionadas porque el factor patógeno es el principal identificado en la sobrecarga crónica, siendo así una de las causas principales del dolor de espalda, radiculopatía y discapacidad.	modelo stacking ensemble permite mejorar la predicción, también ayuda a identificar continuamente las muestras de predicción errónea de las características de los clasificadores de la capa base.		$2.\frac{precission\ .\ recall}{precission + recall}$ (C. Cano & Ruiz, 2018)	Razón
				Área bajo la curva AUC=∫(0 to 1) ROC(x) dx (Martínez Pérez & Pérez Martin, 2023)	
				Especificidad $\frac{\mathrm{TN}}{\mathrm{TN}\ +\ \mathrm{FP}}$ (Dymaxion Labs, 2021)	
				MCC MCC = (TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) (Chicco, Starovoitov, et al., 2021)	
				Perdida (Loss) CrossEntropy=−*n*1∑*i*=1*n* ∑*j*=1*myij*log(*y*^*ij*) (Wang et al., 2022)	

Fuente: Elaboración propia

Anexo 3: Matriz de Consistencia

Tabla Nº 144: Matriz de consistencia

Problema	Objetivo	Hipótesis	Variable	Dimensiones	Indicadores	Metodología
P.G. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir enfermedades de la columna vertebral?	**O.G. A**plicar un sistema inteligente con ensemble machine learning con el método stacking mediante el uso de balanceo de datos que permitirá predecir enfermedades de la columna vertebral	**H.G.** El sistema inteligente con ensemble machine learning con el método stacking mediante el uso de balanceo de datos ayuda y mejora significativamente predecir las enfermedades de la columna vertebral	**Independiente:** Machine learning			
P.E.1. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la exactitud (Accuracy)?	**O.E.1.** aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la exactitud (Accuracy	**H.E.1.** EL sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la exactitud (Accuracy	**Dependiente:** Predecir enfermedades de la columna vertebral	Métricas de evaluación	Exactitud (Acuraccy) $\frac{TP + TN}{TP + TN + FP´FN}$	**Tipo de investigación:** Aplicada **Diseño de investigación:** Experimental de tipo pre-experimental **Metodología:** KDD
P.E.2. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo	**O.E.2.** Aplicar un sistema inteligente con stacking ensemble machine	**H.E.2.** El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos			Precisión $\frac{TP}{TP + FP}$	

de datos permitirá predecir las enfermedades de la columna vertebral en función de la precisión?,	learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la precisión (Precision	mejora la predicción de las enfermedades de la columna vertebral en función de la precisión				
P.E.3. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la exhaustividad (Recall)?	**O.E.3.** Aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la exhaustividad	**H.E.3.** El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la exhaustividad (Recall)			Exhaustividad $\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$	
P.E.4. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la F1-score?	**O.E.4.** Aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la F1-score	**H.E.4.** El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la F1-score			F1-score $2.\frac{precission\ .\ recall}{precission + recall}$	

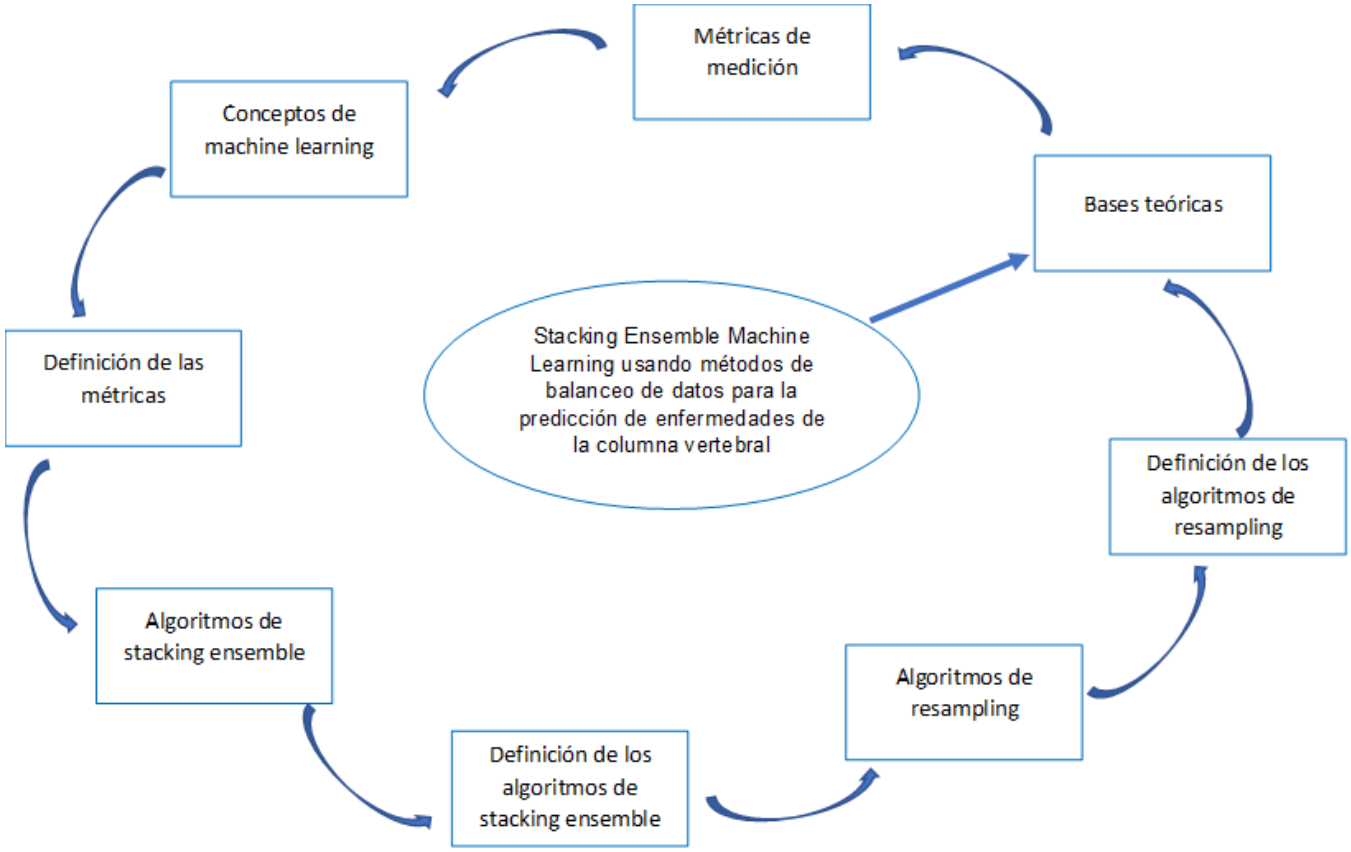
P.E.5. ?,¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función del área bajo la curva ROC (Area Under the ROC Curve – AUC-ROC)?	**O.E.5.** Aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función del área bajo la curva (Area Under the ROC Curve – AUC-ROC	**H.E.5.** El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función del área bajo la curva (Area Under the ROC Curve – AUC-ROC			Área bajo la curva $AUC=\int(0 \text{ to } 1)\ ROC(x)\ dx$	
P.E.6. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la especificidad (Specificity)?,	**O.E.6.** Aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la especificidad	**O.E.6.** El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la especificidad			Especificidad $\frac{TN}{TN + FP}$	
P.E.7. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de	**O.E.7.** Aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de	**O.E.7.** El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de			MCC MCC = (TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	

la columna vertebral en función de MCC (Matthews Correlation Coefficient)?	balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de MCC (Matthews Correlation Coefficient)	la columna vertebral en función MCC (Matthews Correlation Coefficient)				
P.E.8. ¿Cómo un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos permitirá predecir las enfermedades de la columna vertebral en función de la pérdida (Loss)?	**O.E.8.** Aplicar un sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos que permitirá predecir las enfermedades de la columna vertebral en función de la perdida (Loss)	**O.E.8.** El sistema inteligente con stacking ensemble machine learning mediante el uso de balanceo de datos mejora la predicción de las enfermedades de la columna vertebral en función de la perdida (Loss).			**Perdida (Loss)** **CrossEntropy=−*n*1 ∑*i*=1*n* ∑*j*=1*myij* log(*y*^*ij*)**	

Fuente: Elaboración propia

Anexo 4: Marco teórico

Figura Nº 58: Diagrama de Marco teórico

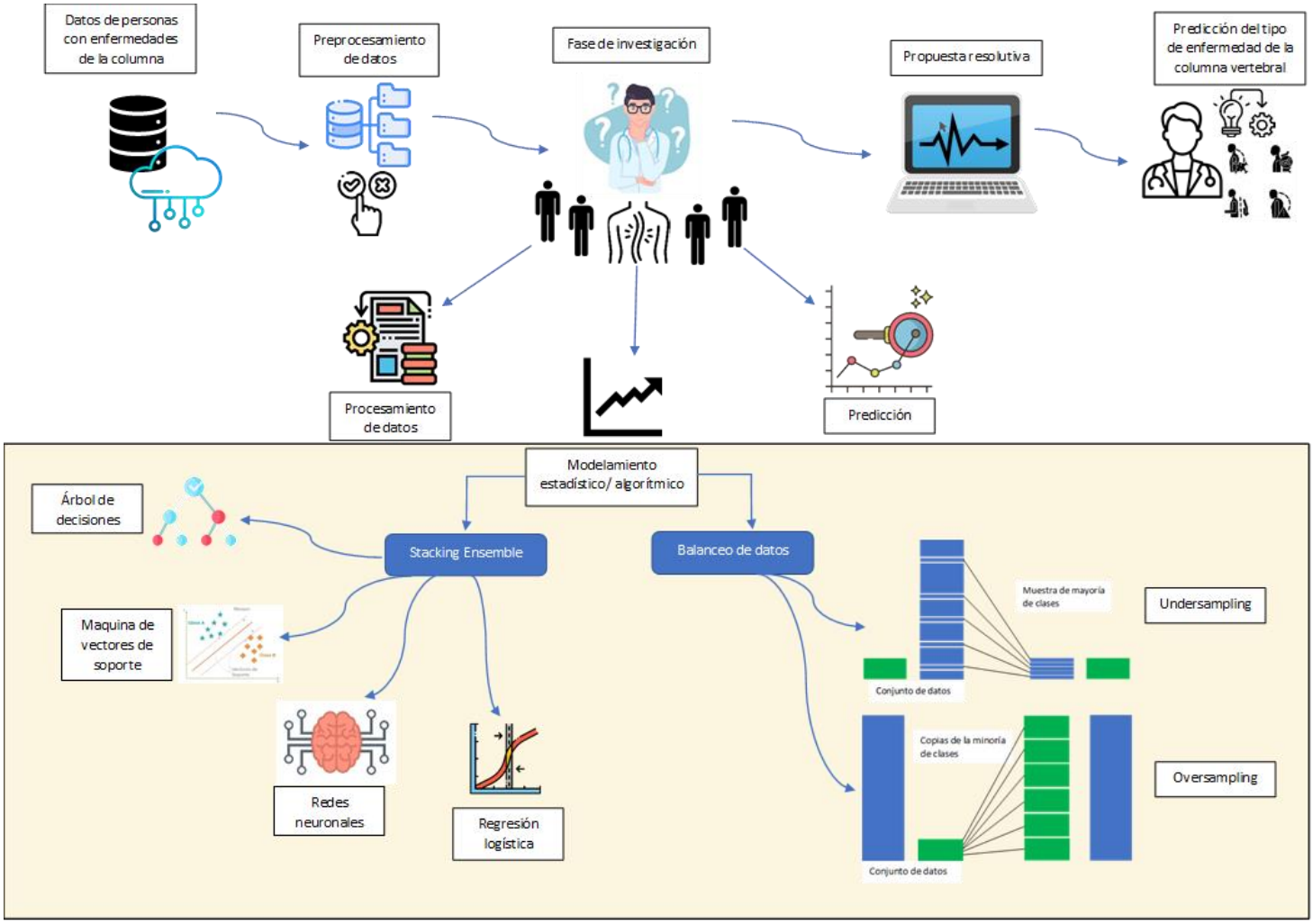


Fuente: Elaboración propia

Anexo 5: Propuesta de solución

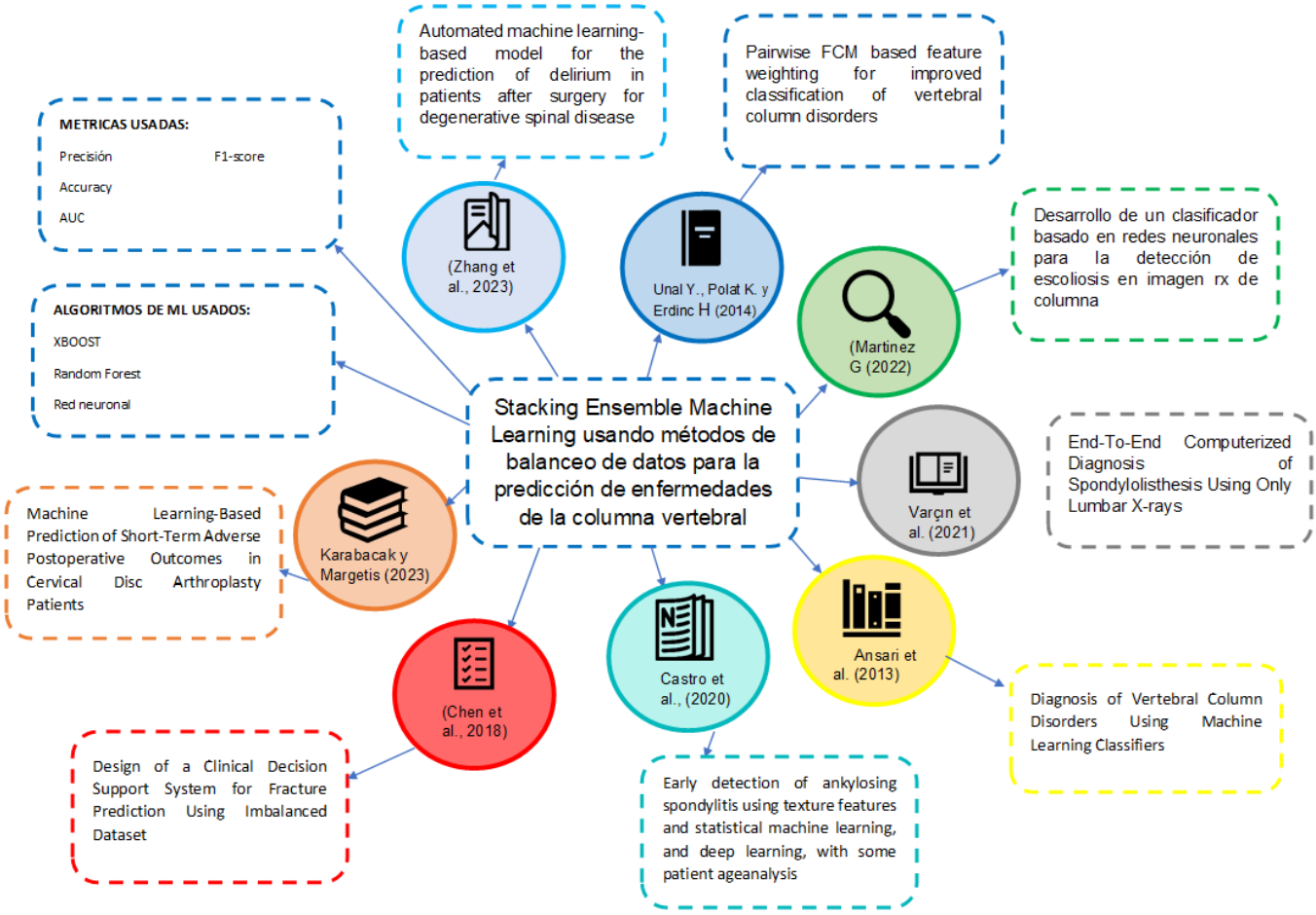
Figura Nº 59: Propuesta de solución

Fuente: Elaboración propia



Anexo 6: Mapa mental de antecedentes

Figura Nº 60: Mapa mental de antecedentes



Fuente: Elaboración propia

Anexo 7:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO DECISION TREE

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Decision Tree

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	82.26%
2	Precisión	Razón	(TP/(TP+FP)) * 100	77.57%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	80.29%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	77.99%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.84
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	72.37%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	93.70%

Alfredo Daza Vergaray

40466240

Anexo 8:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO GRADIENT BOOSTING

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Gradient Boosting

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	77.42%
2	Precisión	Razón	(TP/(TP+FP)) * 100	68.09%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	68.52%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	68.33%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.93
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	63.39%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	90.12%

Alfredo Daza Vergaray

40466240

Anexo 9:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO K-NEAREST NEIGHBORT

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	k-nearest neighbors

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	85.48%
2	Precisión	Razón	(TP/(TP+FP)) * 100	81.61%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	83.80%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	82.50%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.95
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	76.69%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	88.53%

Alfredo Daza Vergaray

40466240

Anexo 10:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO NAIVE BAYES

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Naive Bayes

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	87.10%
2	Precisión	Razón	(TP/(TP+FP)) * 100	83.34%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	82.41%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	82.59%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.95
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	78.96%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	92.95%

Alfredo Daza Vergaray

40466240

Anexo 11:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO NEAREST CENTROID

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Nearest Centroid

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	80.65%
2	Precisión	Razón	(TP/(TP+FP)) * 100	77.16%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	79.28%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	76.34%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	No aplica
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	70.42%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	92.95%

Alfredo Daza Vergaray

40466240

Anexo 12:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO RANDOM FOREST

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Random Forest

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	90.32%
2	Precisión	Razón	(TP/(TP+FP)) * 100	86.40%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	85.88%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	85.41%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.97
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	81.54%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	90.12%

Alfredo Daza Vergaray

40466240

Anexo 13:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO REDES NEURONALES

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Neural network

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	87.10%
2	Precisión	Razón	(TP/(TP+FP)) * 100	82.74%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	62.96%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	56.03%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.90
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	64.77%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	95.14%
8	Perdida	Razón		0.28

Alfredo Daza Vergaray

40466240

Anexo 14:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO REDES NEURONALES CONLOVULSIONALES

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Convolutional neural networks

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	82.26%
2	Precisión	Razón	(TP/(TP+FP)) * 100	75.82%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	63.77%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	55.91%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.92
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	65.89%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	87.09%
8	Perdida	Razón		2.94

Alfredo Daza Vergaray

40466240

Anexo 15:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO REGRESIÓN LOGÍSTICA

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Logistic regression

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	85.48%
2	Precisión	Razón	(TP/(TP+FP)) * 100	79.83%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	81.10%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	80.06%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.97
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	76.74%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	93.70%

Alfredo Daza Vergaray

40466240

Anexo 16:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO MAQUINA DE VECTORES DE SOPORTE

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	Support Vector Machine

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	87.10%
2	Precisión	Razón	(TP/(TP+FP)) * 100	81.48%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	81.48%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	81.48%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.95
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	78.91%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	93.70%

Alfredo Daza Vergaray

40466240

Anexo 17:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO XGBOOST

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	XGBoost

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	88.71%
2	Precisión	Razón	(TP/(TP+FP)) * 100	85.17%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	85.88%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	84.58%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.96
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	81.97%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	94.30%

Alfredo Daza Vergaray

40466240

Anexo 18:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL ALGORITMO ADABOOST

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	30/10/2023
Algoritmo	AdaBoost

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	85.48%
2	Precisión	Razón	(TP/(TP+FP)) * 100	79.83%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	81.10%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	80.06%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	0.87
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	76.74%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	93.70%

Alfredo Daza Vergaray

40466240

Anexo 19:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL MODELO STACKING 1

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	20/10/2023
Algoritmo	Stacking

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	88.7%
2	Precisión	Razón	(TP/(TP+FP)) * 100	84.18%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	85.88%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	84.83%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	44.81%
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	84.69%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	95%

Alfredo Daza Vergaray

40466240

Anexo 20:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL MODELO STACKING 2

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	20/10/2023
Algoritmo	Stacking 2

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	87.09%
2	Precisión	Razón	(TP/(TP+FP)) * 100	82.72%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	84.83%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	83.57%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	48.21%
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	82.04%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	94%

Alfredo Daza Vergaray

40466240

Anexo 21:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL MODELO STACKING 3

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	20/11/2023
Algoritmo	Stacking 3

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	88.7%
2	Precisión	Razón	(TP/(TP+FP)) * 100	84.18%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	85.88%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	84.83%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	44.81%
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	84.69%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	95%

Alfredo Daza Vergaray

40466240

Anexo 22:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL MODELO STACKING 4

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	20/11/2023
Algoritmo	Stacking 4

MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	88.7%
2	Precisión	Razón	(TP/(TP+FP)) * 100	84.18%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	85.88%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	84.83%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	44.81%
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	84.69%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	95%

Alfredo Daza Vergaray

40466240

Anexo 23:

INSTRUMENTO DE FICHA DE REGISTRO PARA EL MODELO STACKING 5

RESUMEN DE EVALUACIÓN	
Tipo de prueba	Post test
Investigador	• Gutierrez Allende Diego Edú
Fecha de inicio	20/11/2023
Algoritmo	Stacking 5

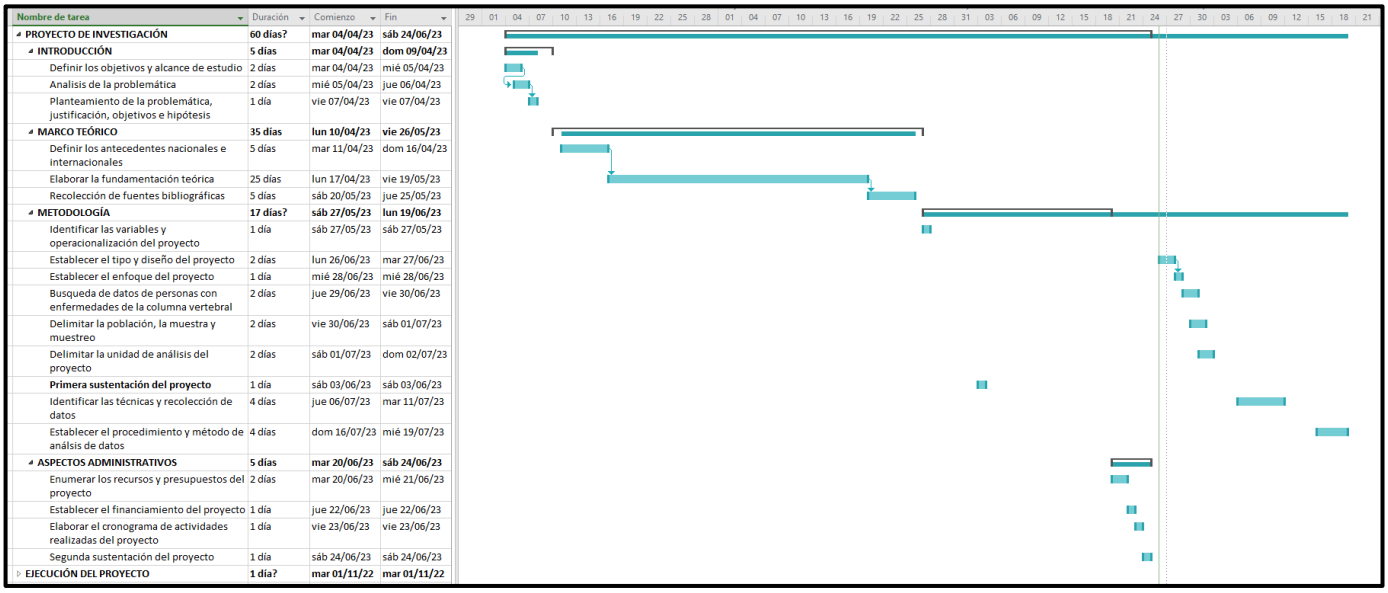
MATRIZ DE CONFUSIÓN			
		Estimación por el modelo	
		Negativo (N)	**Positivo (P)**
Real	**Negativo**	TN	FP
	Positivo	FN	TP

MÉTRICAS POR EVALUAR				
Ítems	**Indicador**	**Medida**	**Fórmula**	**Precisión**
1	Exactitud	Razón	(TP+TN)/(TP+TN+FP+FN)	88.7%
2	Precisión	Razón	(TP/(TP+FP)) * 100	84.18%
3	Sensibilidad	Razón	(TP/(TP+FN)) * 100	85.88%
4	F1-Score	Razón	2 * ((Precisión * Sensibilidad) / (Precisión + Sensibilidad))	84.83%
5	AUC - ROC	Razón	∑[(FP[i+1] - FP[i]) * (TP[i] + TP[i+1]) / 2]	44.81%
6	MCC	Razón	(TP * TN - FP * FN) / √((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))	84.69%
7	Especificidad	Razón	(TN/(TN+FP)) * 100	95%

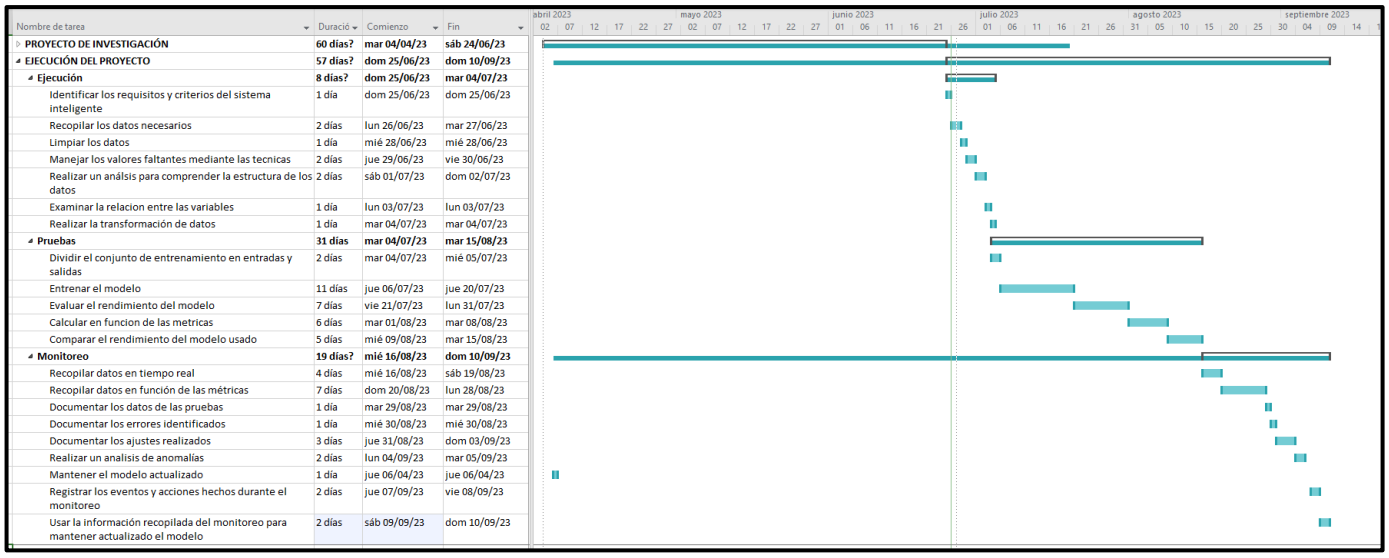
Alfredo Daza Vergaray

40466240

Anexo 17: Diagrama de Gantt



Nombre de tarea	Duración	Comienzo	Fin
PROYECTO DE INVESTIGACIÓN	**60 días?**	**mar 04/04/23**	**sáb 24/06/23**
INTRODUCCIÓN	**5 días**	**mar 04/04/23**	**dom 09/04/23**
Definir los objetivos y alcance de estudio	2 días	mar 04/04/23	mié 05/04/23
Analisis de la problemática	2 días	mié 05/04/23	jue 06/04/23
Planteamiento de la problemática, justificación, objetivos e hipótesis	1 día	vie 07/04/23	vie 07/04/23
MARCO TEÓRICO	**35 días**	**lun 10/04/23**	**vie 26/05/23**
Definir los antecedentes nacionales e internacionales	5 días	mar 11/04/23	dom 16/04/23
Elaborar la fundamentación teórica	25 días	lun 17/04/23	vie 19/05/23
Recolección de fuentes bibliográficas	5 días	sáb 20/05/23	jue 25/05/23
METODOLOGÍA	**17 días?**	**sáb 27/05/23**	**lun 19/06/23**
Identificar las variables y operacionalización del proyecto	1 día	sáb 27/05/23	sáb 27/05/23
Establecer el tipo y diseño del proyecto	2 días	lun 26/06/23	mar 27/06/23
Establecer el enfoque del proyecto	1 día	mié 28/06/23	mié 28/06/23
Busqueda de datos de personas con enfermedades de la columna vertebral	2 días	jue 29/06/23	vie 30/06/23
Delimitar la población, la muestra y muestreo	2 días	vie 30/06/23	sáb 01/07/23
Delimitar la unidad de análisis del proyecto	2 días	sáb 01/07/23	dom 02/07/23
Primera sustentación del proyecto	1 día	sáb 03/06/23	sáb 03/06/23
Identificar las técnicas y recolección de datos	4 días	jue 06/07/23	mar 11/07/23
Establecer el procedimiento y método de análsis de datos	4 días	dom 16/07/23	mié 19/07/23
ASPECTOS ADMINISTRATIVOS	**5 días**	**mar 20/06/23**	**sáb 24/06/23**
Enumerar los recursos y presupuestos del proyecto	2 días	mar 20/06/23	mié 21/06/23
Establecer el financiamiento del proyecto	1 día	jue 22/06/23	jue 22/06/23
Elaborar el cronograma de actividades realizadas del proyecto	1 día	vie 23/06/23	vie 23/06/23
Segunda sustentación del proyecto	1 día	sáb 24/06/23	sáb 24/06/23
EJECUCIÓN DEL PROYECTO	**1 día?**	**mar 01/11/22**	**mar 01/11/22**

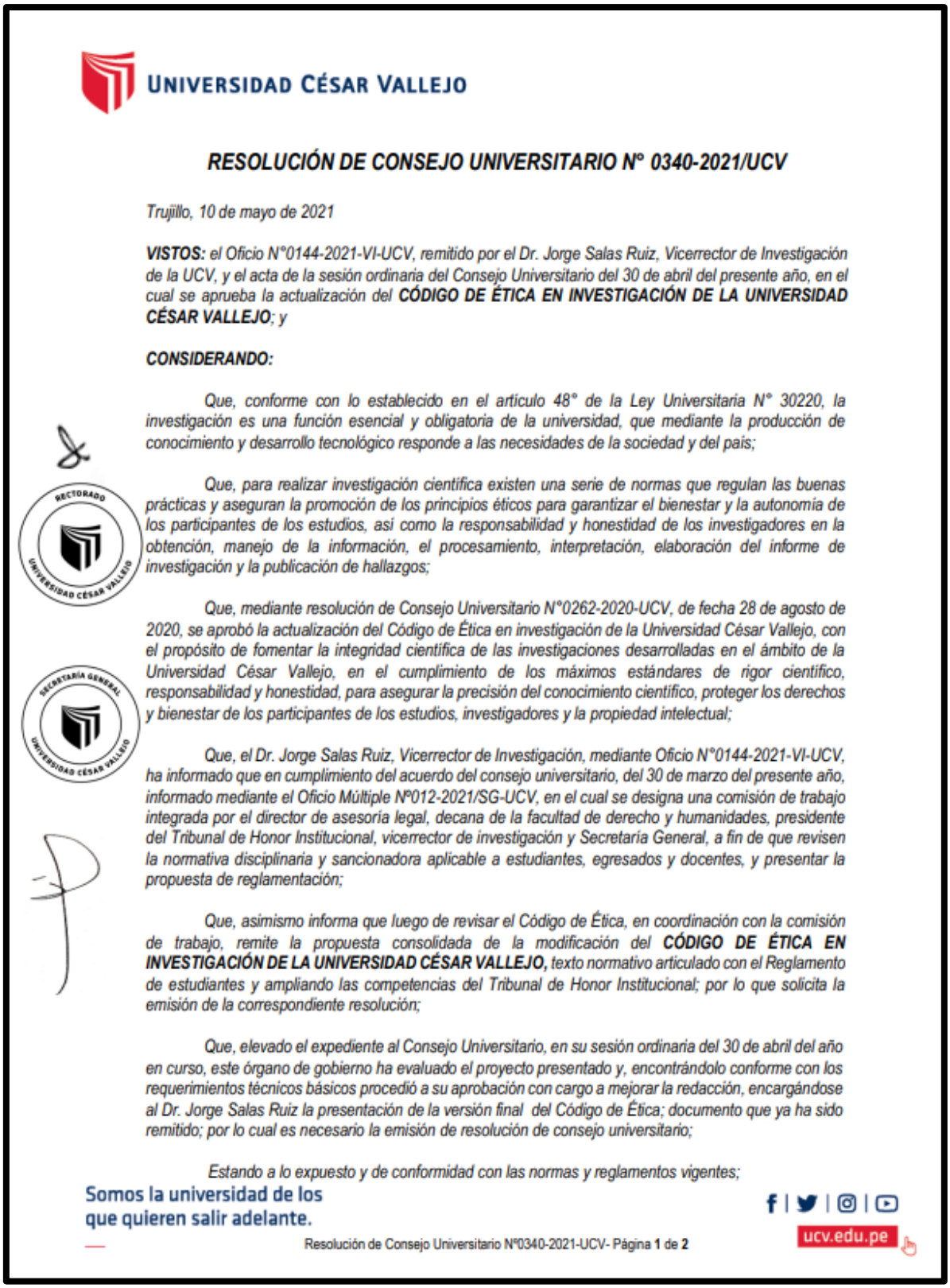


Nombre de tarea	Duració	Comienzo	Fin
PROYECTO DE INVESTIGACIÓN	**60 días?**	**mar 04/04/23**	**sáb 24/06/23**
EJECUCIÓN DEL PROYECTO	**57 días?**	**dom 25/06/23**	**dom 10/09/23**
Ejecución	**8 días?**	**dom 25/06/23**	**mar 04/07/23**
Identificar los requisitos y criterios del sistema inteligente	1 día	dom 25/06/23	dom 25/06/23
Recopilar los datos necesarios	2 días	lun 26/06/23	mar 27/06/23
Limpiar los datos	1 día	mié 28/06/23	mié 28/06/23
Manejar los valores faltantes mediante las tecnicas	2 días	jue 29/06/23	vie 30/06/23
Realizar un análsis para comprender la estructura de los datos	2 días	sáb 01/07/23	dom 02/07/23
Examinar la relacion entre las variables	1 día	lun 03/07/23	lun 03/07/23
Realizar la transformación de datos	1 día	mar 04/07/23	mar 04/07/23
Pruebas	**31 días**	**mar 04/07/23**	**mar 15/08/23**
Dividir el conjunto de entrenamiento en entradas y salidas	2 días	mar 04/07/23	mié 05/07/23
Entrenar el modelo	11 días	jue 06/07/23	jue 20/07/23
Evaluar el rendimiento del modelo	7 días	vie 21/07/23	lun 31/07/23
Calcular en funcion de las metricas	6 días	mar 01/08/23	mar 08/08/23
Comparar el rendimiento del modelo usado	5 días	mié 09/08/23	mar 15/08/23
Monitoreo	**19 días?**	**mié 16/08/23**	**dom 10/09/23**
Recopilar datos en tiempo real	4 días	mié 16/08/23	sáb 19/08/23
Recopilar datos en función de las métricas	7 días	dom 20/08/23	lun 28/08/23
Documentar los datos de las pruebas	1 día	mar 29/08/23	mar 29/08/23
Documentar los errores identificados	1 día	mié 30/08/23	mié 30/08/23
Documentar los ajustes realizados	3 días	jue 31/08/23	dom 03/09/23
Realizar un analisis de anomalías	2 días	lun 04/09/23	mar 05/09/23
Mantener el modelo actualizado	1 día	jue 06/04/23	jue 06/04/23
Registrar los eventos y acciones hechos durante el monitoreo	2 días	jue 07/09/23	vie 08/09/23
Usar la información recopilada del monitoreo para mantener actualizado el modelo	2 días	sáb 09/09/23	dom 10/09/23

Figura Nº 61: Diagrama de Gantt

Fuente: Elaboración propia

Anexo 18: Resolución de consejo Universitario



UNIVERSIDAD CÉSAR VALLEJO

RESOLUCIÓN DE CONSEJO UNIVERSITARIO N° 0340-2021/UCV

Trujillo, 10 de mayo de 2021

VISTOS: *el Oficio N°0144-2021-VI-UCV, remitido por el Dr. Jorge Salas Ruiz, Vicerrector de Investigación de la UCV, y el acta de la sesión ordinaria del Consejo Universitario del 30 de abril del presente año, en el cual se aprueba la actualización del* ***CÓDIGO DE ÉTICA EN INVESTIGACIÓN DE LA UNIVERSIDAD CÉSAR VALLEJO****; y*

CONSIDERANDO:

Que, conforme con lo establecido en el artículo 48° de la Ley Universitaria N° 30220, la investigación es una función esencial y obligatoria de la universidad, que mediante la producción de conocimiento y desarrollo tecnológico responde a las necesidades de la sociedad y del país;

Que, para realizar investigación científica existen una serie de normas que regulan las buenas prácticas y aseguran la promoción de los principios éticos para garantizar el bienestar y la autonomía de los participantes de los estudios, así como la responsabilidad y honestidad de los investigadores en la obtención, manejo de la información, el procesamiento, interpretación, elaboración del informe de investigación y la publicación de hallazgos;

Que, mediante resolución de Consejo Universitario N°0262-2020-UCV, de fecha 28 de agosto de 2020, se aprobó la actualización del Código de Ética en investigación de la Universidad César Vallejo, con el propósito de fomentar la integridad científica de las investigaciones desarrolladas en el ámbito de la Universidad César Vallejo, en el cumplimiento de los máximos estándares de rigor científico, responsabilidad y honestidad, para asegurar la precisión del conocimiento científico, proteger los derechos y bienestar de los participantes de los estudios, investigadores y la propiedad intelectual;

Que, el Dr. Jorge Salas Ruiz, Vicerrector de Investigación, mediante Oficio N°0144-2021-VI-UCV, ha informado que en cumplimiento del acuerdo del consejo universitario, del 30 de marzo del presente año, informado mediante el Oficio Múltiple N°012-2021/SG-UCV, en el cual se designa una comisión de trabajo integrada por el director de asesoría legal, decana de la facultad de derecho y humanidades, presidente del Tribunal de Honor Institucional, vicerrector de investigación y Secretaría General, a fin de que revisen la normativa disciplinaria y sancionadora aplicable a estudiantes, egresados y docentes, y presentar la propuesta de reglamentación;

Que, asimismo informa que luego de revisar el Código de Ética, en coordinación con la comisión de trabajo, remite la propuesta consolidada de la modificación del ***CÓDIGO DE ÉTICA EN INVESTIGACIÓN DE LA UNIVERSIDAD CÉSAR VALLEJO,*** *texto normativo articulado con el Reglamento de estudiantes y ampliando las competencias del Tribunal de Honor Institucional; por lo que solicita la emisión de la correspondiente resolución;*

Que, elevado el expediente al Consejo Universitario, en su sesión ordinaria del 30 de abril del año en curso, este órgano de gobierno ha evaluado el proyecto presentado y, encontrándolo conforme con los requerimientos técnicos básicos procedió a su aprobación con cargo a mejorar la redacción, encargándose al Dr. Jorge Salas Ruiz la presentación de la versión final del Código de Ética; documento que ya ha sido remitido; por lo cual es necesario la emisión de resolución de consejo universitario;

Estando a lo expuesto y de conformidad con las normas y reglamentos vigentes;

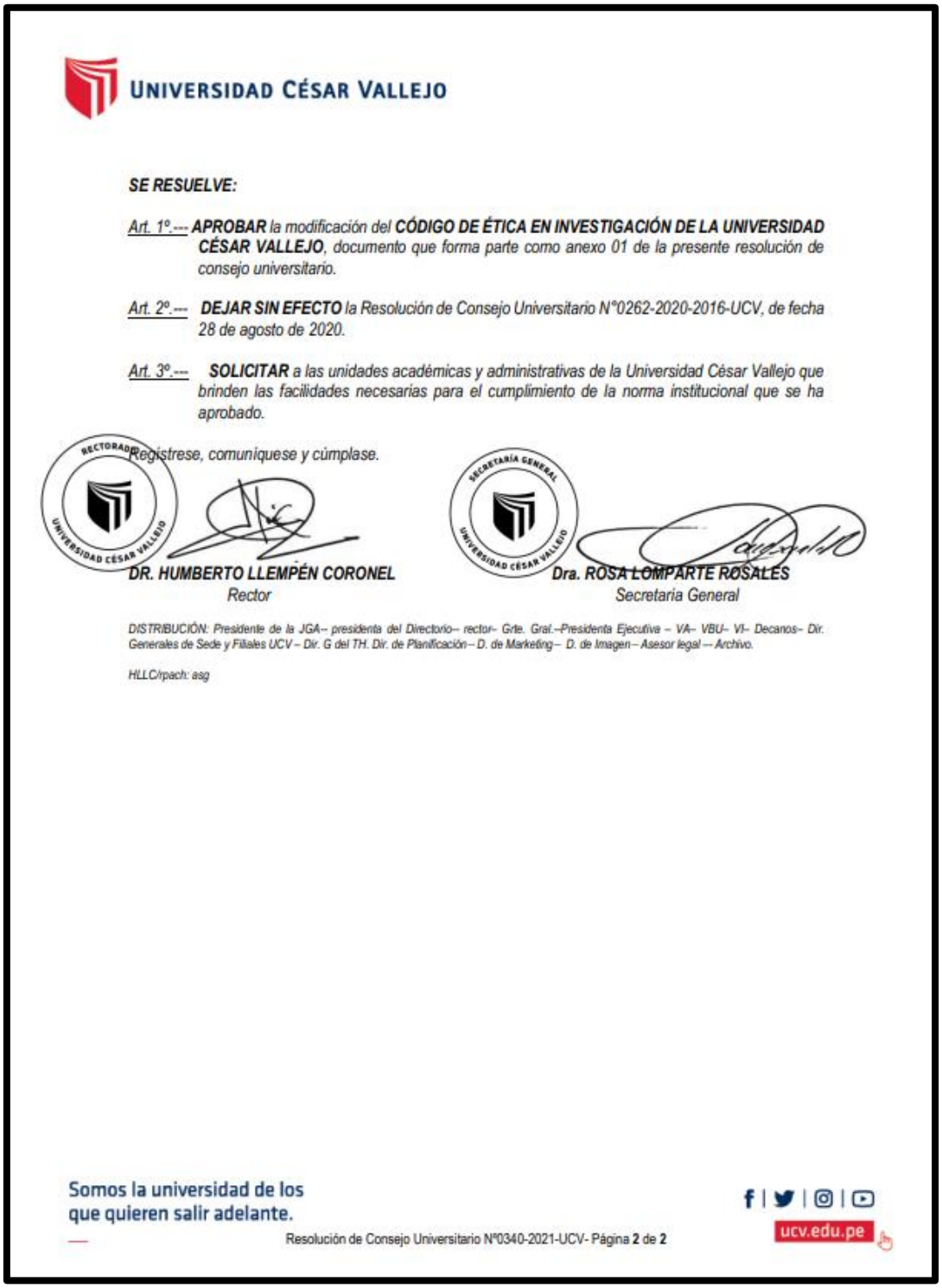
Somos la universidad de los que quieren salir adelante.

ucv.edu.pe

Figura Nº 62: Resolución de consejo universitario

Fuente: UCV

Anexo 18: Resolución de consejo Universitario



UNIVERSIDAD CÉSAR VALLEJO

SE RESUELVE:

Art. 1°.--- **APROBAR** la modificación del **CÓDIGO DE ÉTICA EN INVESTIGACIÓN DE LA UNIVERSIDAD CÉSAR VALLEJO**, documento que forma parte como anexo 01 de la presente resolución de consejo universitario.

Art. 2°.--- **DEJAR SIN EFECTO** la Resolución de Consejo Universitario N°0262-2020-2016-UCV, de fecha 28 de agosto de 2020.

Art. 3°.--- **SOLICITAR** a las unidades académicas y administrativas de la Universidad César Vallejo que brinden las facilidades necesarias para el cumplimiento de la norma institucional que se ha aprobado.

Regístrese, comuníquese y cúmplase.

DR. HUMBERTO LLEMPÉN CORONEL
Rector

Dra. ROSA LOMPARTE ROSALES
Secretaria General

DISTRIBUCIÓN: Presidente de la JGA– presidenta del Directorio– rector– Grte. Gral.–Presidenta Ejecutiva – VA– VBU– VI– Decanos– Dir. Generales de Sede y Filiales UCV – Dir. G del TH. Dir. de Planificación – D. de Marketing – D. de Imagen – Asesor legal – Archivo.

HLLC/rpach: asg

Somos la universidad de los que quieren salir adelante.

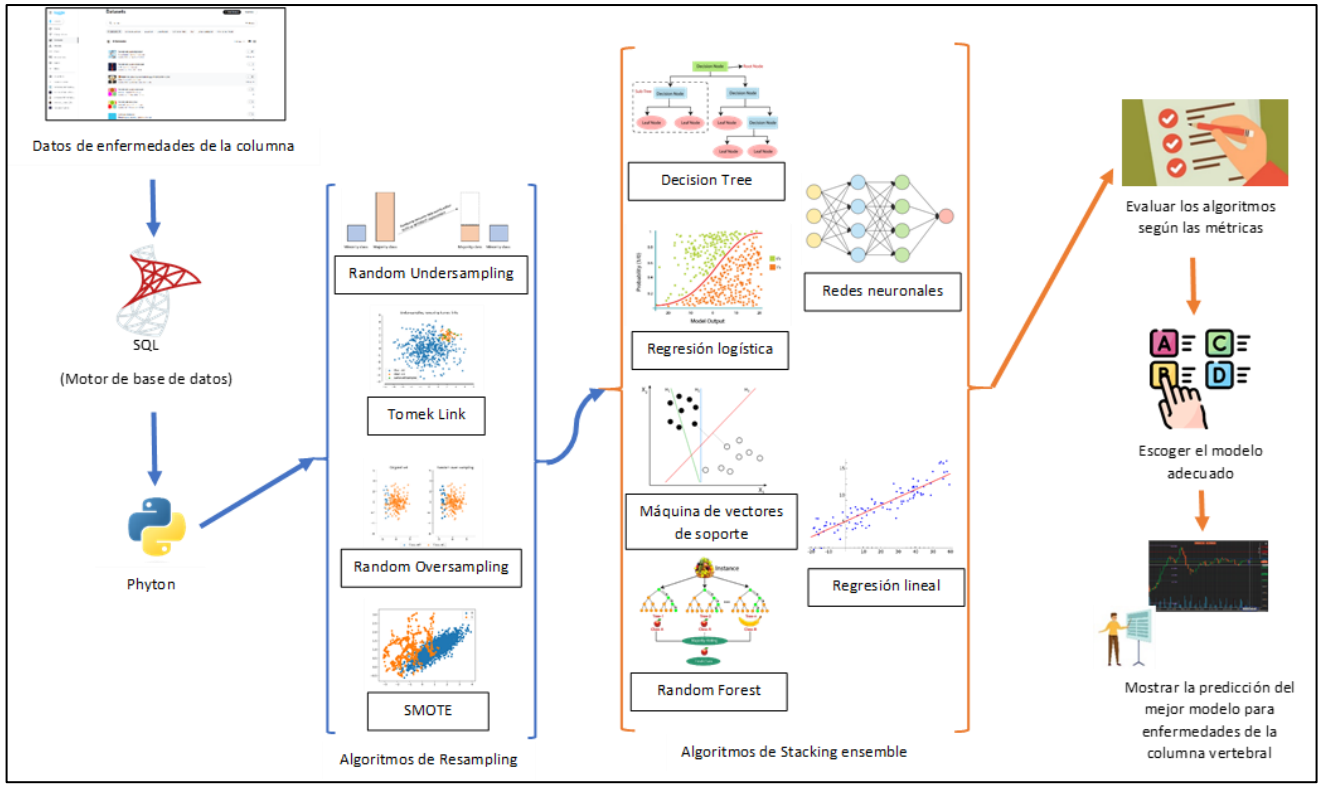
Resolución de Consejo Universitario N°0340-2021-UCV- Página 2 de 2

ucv.edu.pe

Figura Nº 62: Resolución de consejo universitario

Fuente: UCV

Figura Nº 63: Prototipo del sistema inteligente



Fuente: Elaboración propia

Anexo 20: Innovación y aporte tecnológico

Antecedentes	Sistema Inteligente	Técnicas usadas (algoritmos)	Validación cruzada	Selección de variables	Lenguaje/ plataforma de programación o biblioteca	Métricas	Metodología de desarrollo
1	No	Redes neuronales convolucionales	No especifica	No especifica	Colab Python	Accuracy Matriz de confusión	No especifica
2	Sí	XGBOOST Random Forest AdaBoost GNB CNB MLP SVM Regresión logística	No especifica	No especifica	Python	AUROC Sensivity Accuracy Specificity F1-Score Youden Index	No especifica
3	No	Redes Neuronales Convolucionales	No especifica	No especifica	No especifica	Accuracy Recall Specificity	No especifica

4	No	-Red neuronal de retropropagación -Red neuronal de regresión -Máquina de vectores de soporte	Si se aplicó	No especifica	No especifica	Acuraccy	No especifica
5	No	Decision tree Random Forest	Si se aplicó	Si se aplicó	Python	Acuraccy F1-score Recall Precision	No especifica
6	No	Algoritmo genérico integrado Maquina de vectores de soporte Undersampling SMOTE Decision tree	No especifica	No especifica	No especifica	AUC Recall Specificity G-mean Acuraccy	No especifica
7	Si	Random Forest XGBoost LigthGBM	No especifica	No especifica	No especifica	AUC ROC Precisión Recall	No especifica

		CatBoost				Acuraccy MCC	
8	No	Perceptrón multicapa KNN SVM	No especifica	No especifica	No especifica	Precision Recall Specificity AUC ROC	No especifica

Tabla Nº 145: Innovación y aporte tecnológico

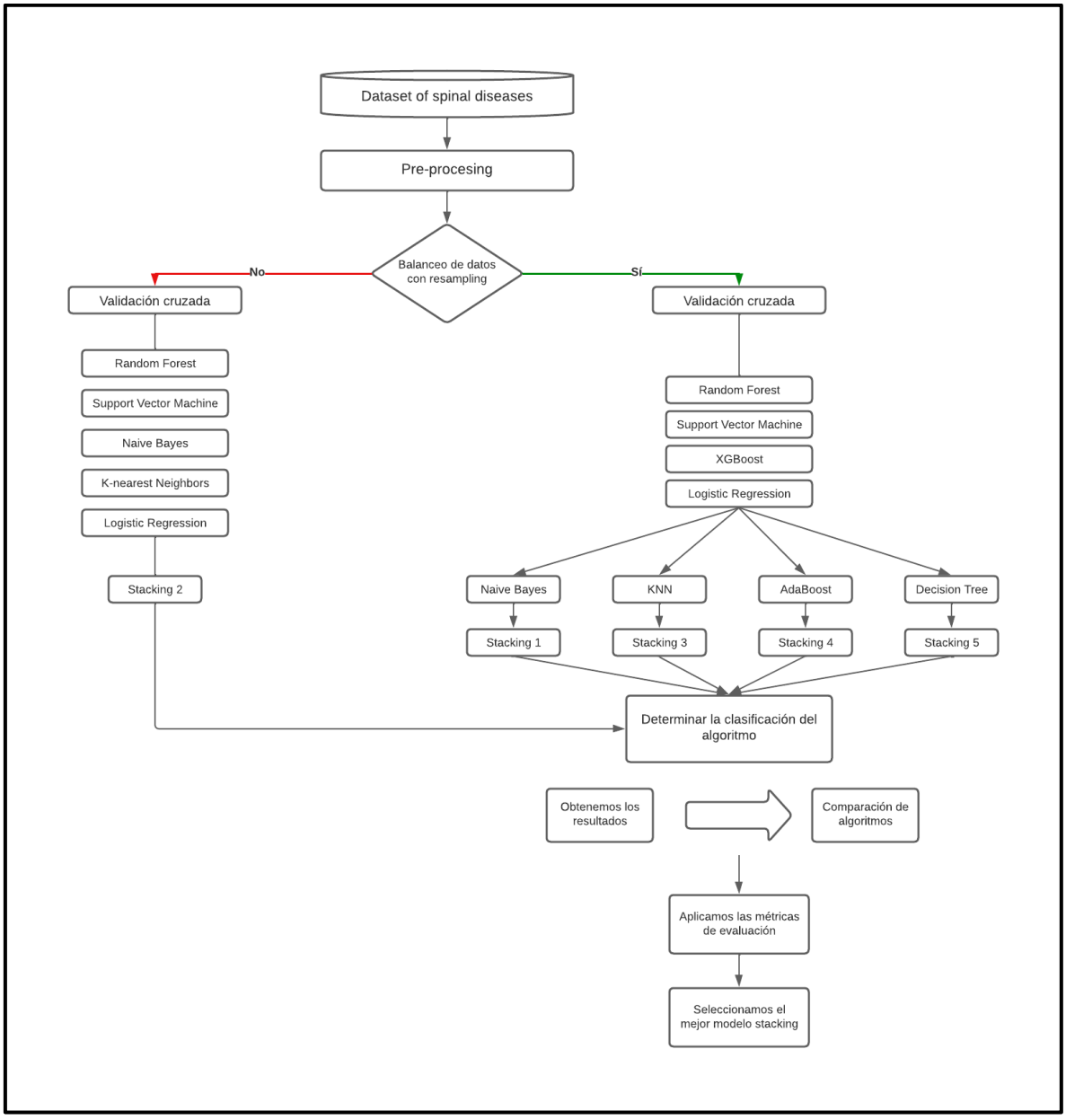
Fuente: Elaboración propia

En esta investigación se plantea realizar un sistema inteligente con machine learning haciendo uso de los métodos stacking ensemble y balanceo de datos (resampling), mediante el planteamiento de la metodología KDD. Teniendo en cuenta que otros investigadores no hicieron uso de estos dos métodos y solo hicieron uso del método resampling para tener los conjuntos de datos balanceados como se muestra en uno de los artículos de la revisión de literatura realizada. También se tomará en cuenta algunos puntos que han realizado otros investigadores, como el uso de otros algoritmos que obtuvieron una buena precisión y algunas otras métricas de evaluación de modelo.

Para desarrollar este proyecto de investigación, se hará uso de distintos softwares con un vínculo mayoritario en la aplicación de ciencia de datos, como el software de Anaconda y Jupyter para el desarrollo de los modelos, como lenguaje de programación se usará Python y el software Pycharm para el diseño del sistema inteligente. Se usará la metodología KDD, que estará compuesta por las siguientes fases: Problema, entender el dominio y definir las metas; integración y recopilación; selección, limpieza y transformación; Minería de datos (Modelado), Evaluación e interpretación, Difusión y uso.

Anexo 21:

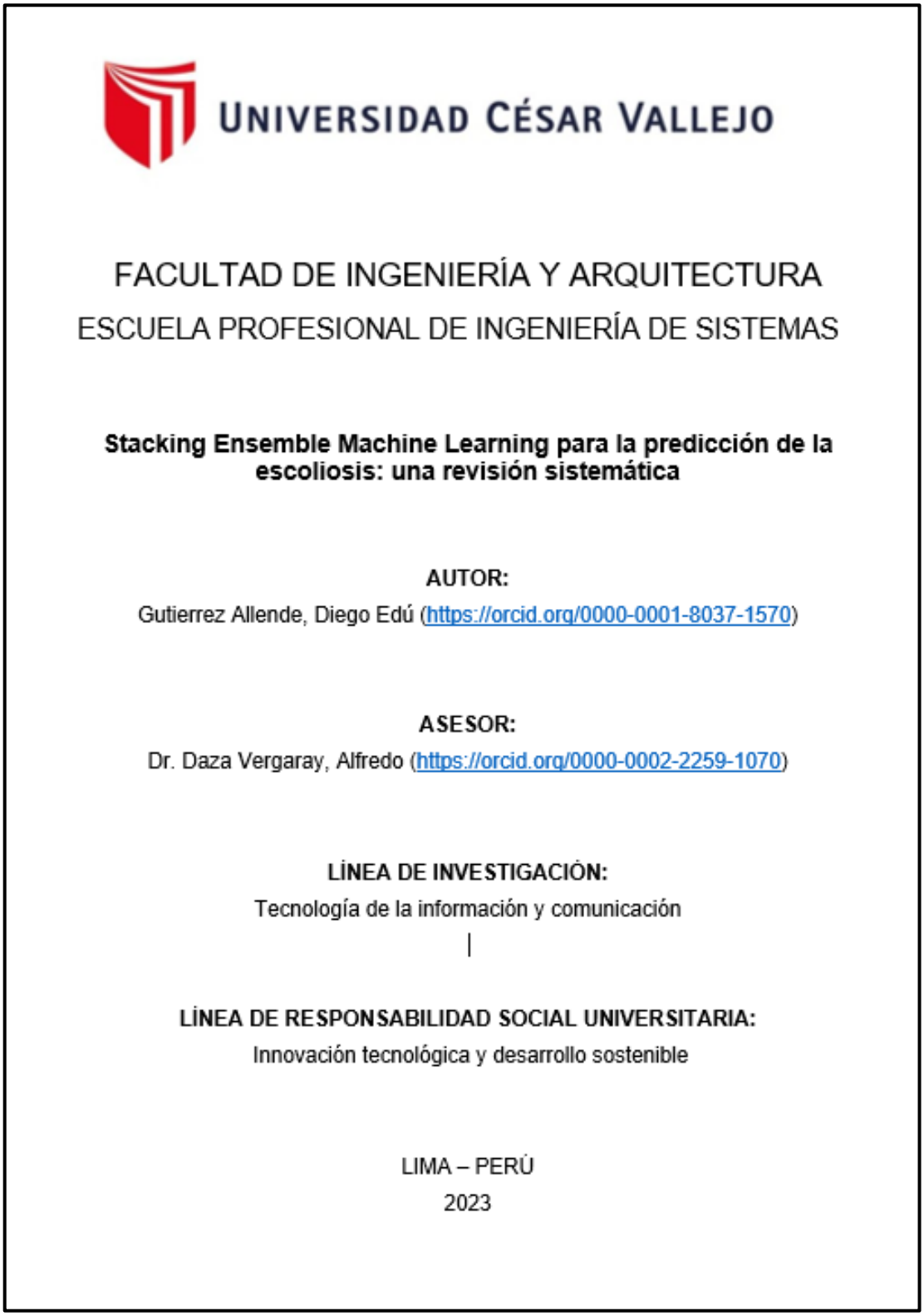
Figura Nº 64: Prototipo del algoritmo



Fuente: Elaboración propia

Anexo 22:

Figura Nº 65 Artículo de revisión de Literatura



UNIVERSIDAD CÉSAR VALLEJO

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Stacking Ensemble Machine Learning para la predicción de la escoliosis: una revisión sistemática

AUTOR:

Gutierrez Allende, Diego Edú (https://orcid.org/0000-0001-8037-1570)

ASESOR:

Dr. Daza Vergaray, Alfredo (https://orcid.org/0000-0002-2259-1070)

LÍNEA DE INVESTIGACIÓN:

Tecnología de la información y comunicación

LÍNEA DE RESPONSABILIDAD SOCIAL UNIVERSITARIA:

Innovación tecnológica y desarrollo sostenible

LIMA – PERÚ

2023

Fuente: Elaboración propia

Desarrollo del Sistema Inteligente con Machine Learning basado en selección de variables y métodos de resampling para predecir las enfermedades de la columna vertebral

En esta sección se da a conocer las actividades que fueron aplicadas según la metodología KDD para el desarrollo de la propuesta tecnológica, ilustrada en el ANEXO 6, y el prototipo en el ANEXO 27. Dicha metodología, se dividió en 5 etapas, estas son las siguientes: selección de datos, preprocesamiento, transformación de los datos, minería de datos e interpretación y evaluación. A continuación, se detalla lo elaborado en cada etapa:

Primera etapa: Selección de datos

El conjunto de datos fue recopilado de la población de 310 personas con enfermedades de la columna vertebral (hernias o espondilolistesis) y personas normales que no padecen de alguna enfermedad de la columna vertebral.

Luego de obtener el conjunto de datos, se detalló en la Tabla Nº 2 las variables, la descripción de cada variable y el rango de los datos en función de cada variable.

Segunda etapa: Preprocesamiento

En esta etapa se procedió a convertir los datos de formato xlsx, hacia el gestor de datos SQL.

Figura Nº 66: BD en Excel de enfermedades de la columna vertebral

pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
63.027817	22.552586	39.609117	40.475232	98.672917	-0.2544	Hernia
39.056951	10.060991	25.015378	28.99596	114.405425	4.564259	Hernia
68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	Hernia
69.297008	24.652878	44.311238	44.64413	101.868495	11.211523	Hernia
49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	Hernia
40.2502	13.921907	25.12495	26.328293	130.327871	2.230652	Hernia
53.432928	15.864336	37.165934	37.568592	120.567523	5.988551	Hernia
45.366754	10.755611	29.038349	34.611142	117.270067	-10.675871	Hernia
43.79019	13.533753	42.690814	30.256437	125.002893	13.289018	Hernia
36.686353	5.010884	41.948751	31.675469	84.241415	0.664437	Hernia
49.70661	13.040974	31.3345	36.665635	108.648265	-7.825986	Hernia
31.232387	17.715819	15.5	13.516568	120.055399	0.499751	Hernia
48.915551	19.964556	40.263794	28.950995	119.321358	8.028895	Hernia
53.57217	20.460828	33.1	33.111342	110.966698	7.044803	Hernia
57.300227	24.188885	47	33.111342	116.806587	5.766947	Hernia

Fuente: Elaboración propia

Figura Nº 67: BD en SQL de enfermedades de la columna vertebral

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
1	63.027817	22.552586	39.609117	40.475232	98.672917	-0.2544	Hernia
2	39.056951	10.060991	25.015378	28.99596	114.405425	4.564259	Hernia
3	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	Hernia
4	69.297008	24.652878	44.311238	44.64413	101.868495	11.211523	Hernia
5	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	Hernia
6	40.2502	13.921907	25.12495	26.328293	130.327871	2.230652	Hernia
7	53.432928	15.864336	37.165934	37.568592	120.567523	5.988551	Hernia
8	45.366754	10.755611	29.038349	34.611142	117.270067	-10.675871	Hernia
9	43.79019	13.533753	42.690814	30.256437	125.002893	13.289018	Hernia
10	36.686353	5.010884	41.948751	31.675469	84.241415	0.664437	Hernia
11	49.70661	13.040974	31.3345	36.665635	108.648265	-7.825986	Hernia
12	31.232387	17.715819	15.5	13.516568	120.055399	0.499751	Hernia
13	48.915551	19.964556	40.263794	28.950995	119.321358	8.028895	Hernia

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
1	63.027817	22.552586	39.609117	40.475232	98.672917	-0.2544	0
2	39.056951	10.060991	25.015378	28.99596	114.405425	4.564259	0
3	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	0
4	69.297008	24.652878	44.311238	44.64413	101.868495	11.211523	0
5	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	0
6	40.2502	13.921907	25.12495	26.328293	130.327871	2.230652	0
7	53.432928	15.864336	37.165934	37.568592	120.567523	5.988551	0
8	45.366754	10.755611	29.038349	34.611142	117.270067	-10.675871	0
9	43.79019	13.533753	42.690814	30.256437	125.002893	13.289018	0
10	36.686353	5.010884	41.948751	31.675469	84.241415	0.664437	0
11	49.70661	13.040974	31.3345	36.665635	108.648265	-7.825986	0
12	31.232387	17.715819	15.5	13.516568	120.055399	0.499751	0
13	48.915551	19.964556	40.263794	28.950995	119.321358	8.028895	0
14	53.57217	20.460828	33.1	33.111342	110.966698	7.044803	0
15	57.300227	24.188885	47	33.111342	116.806587	5.766947	0

Fuente: Elaboración propia

Tercera etapa: Transformación de los datos

Luego de almacenar el conjunto de datos en un documento Excel, se procedió a asignarles un valor numérico como se menciona en la Tabla Nº 4.

Continuando con el desarrollo, se inició el uso de las herramientas de Anaconda y Jupyter Notebook para realizar el análisis exploratorio. Como vista general, se creó una gráfica que incluye todas las variables.

Primeramente, se realizó la visualización de los datos, en donde se observa las primeras 5 filas y las variables principales con las que cuenta la base de datos.

Figura Nº 68: Vista de cabecera y datos iniciales

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254400	Hernia
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	Hernia
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	Hernia
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	Hernia
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	Hernia

Fuente: Elaboración propia

También se realizó una vista de los datos finales de la base de datos

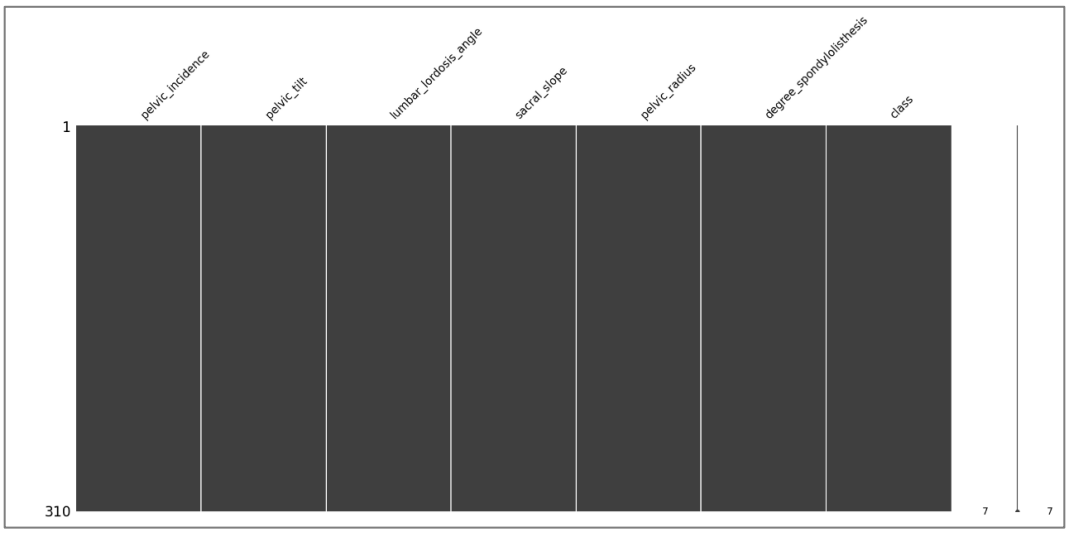
Figura Nº 69: Vista de cabecera y datos finales

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
305	47.903565	13.616688	36.000000	34.286877	117.449062	-4.245395	Normal
306	53.936748	20.721496	29.220534	33.215251	114.365845	-0.421010	Normal
307	61.446597	22.694968	46.170347	38.751628	125.670725	-2.707880	Normal
308	45.252792	8.693157	41.583126	36.559635	118.545842	0.214750	Normal
309	33.841641	5.073991	36.641233	28.767649	123.945244	-0.199249	Normal

Fuente: Elaboración propia

Mediante la vista de las variables con los que cuenta la base de datos. Se procedió a realizar la instalación de la librería missigno, para poder conocer si existen datos faltantes en la base de datos.

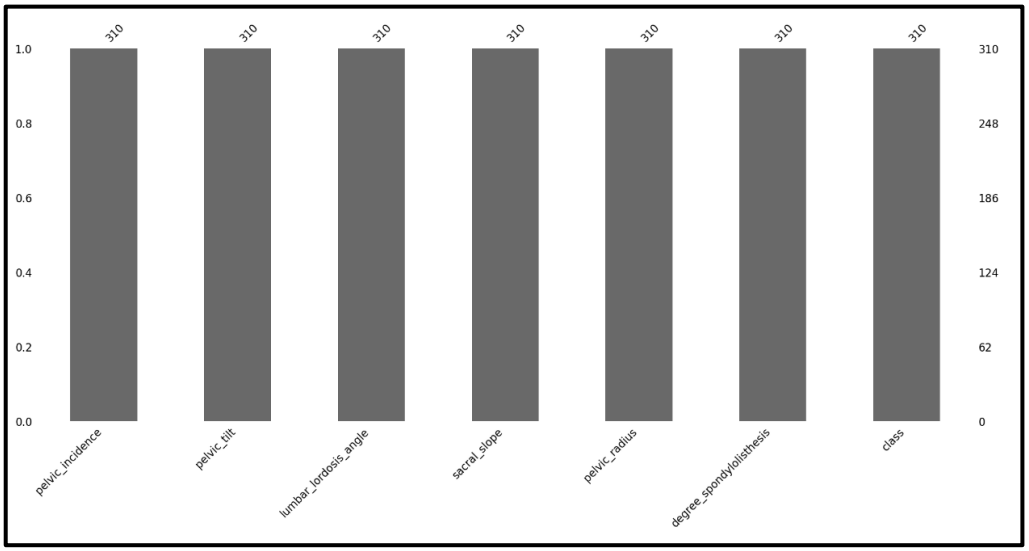
Figura Nº 70: Grafica de barras de datos faltantes



Fuente: Elaboración propia

Mediante esta grafica donde las barras completamente negras, representan que no existe ningún dato faltante en la base de datos. En caso contrario, si hubiese valores faltantes se generarían espacios en blanco en las barras, según las variables. En esta grafica se observa las 7 columnas, la numeración de datos (1-310) en la parte izquierda de la gráfica.

Figura Nº 71: Grafica de barras de datos faltantes por cada variable



Fuente: Elaboración propia

La siguiente grafica también representa a los valores faltantes que puede tener la base de datos, teniendo los campos de las 7 variables en la parte de abajo y la numeración.

Se realizo el uso de distintos códigos para poder verificar que no haya datos faltantes.

Figura Nº 72: Columnas que contengan datos faltantes

```
data.isnull().any()

pelvic_incidence           False
pelvic_tilt                False
lumbar_lordosis_angle      False
sacral_slope               False
pelvic_radius              False
degree_spondylolisthesis   False
class                      False
dtype: bool
```

Fuente: Elaboración propia

Mediante el resultado de ese código se da a conocer si las columnas contienen algún dato faltante, en donde False representa que no hay datos faltantes y True representa que si hay datos faltantes.

Figura Nº 73: Columnas con datos faltantes (missing)

```
null_columns=data.columns[data.isnull().any()]
print(null_columns)
print("No existen datos faltantes")
```

```
Index([], dtype='object')
No existen datos faltantes
```

Fuente: Elaboración propia

Mediante este código, se visualizará los datos faltantes y en caso de que no existan datos faltantes, se generara el mensaje “No existen datos faltantes”.

Figura Nº 74: Numero de datos faltantes por columna

```
data.isnull().sum()
```

```
pelvic_incidence            0
pelvic_tilt                 0
lumbar_lordosis_angle       0
sacral_slope                0
pelvic_radius               0
degree_spondylolisthesis    0
class                       0
dtype: int64
```

Fuente: Elaboración propia

Mediante este código se permitió dar a conocer el número de datos faltantes por columna. Indicando la numeración de 0 si no existe algún valor faltante.

Figura Nº 75: Número de columnas originales y finales

IMPRIMIR NÚMERO DE COLUMNAS ORIGINALES - Y COLUMNAS FINALES

```
# Obtener las columnas iniciales y finales
print("Columnas en el conjunto de datos original: %d" % data.shape[1])
print("Columnas finales: %d" % Columns_with_na_dropped.shape[1])
```

```
Columnas en el conjunto de datos original: 7
Columnas finales: 7
```

```
# porcentaje de columnas que no tengan missing
print(str(Columns_with_na_dropped.shape[1]/data.shape[1]*100) + "%")
```

```
100.0%
```

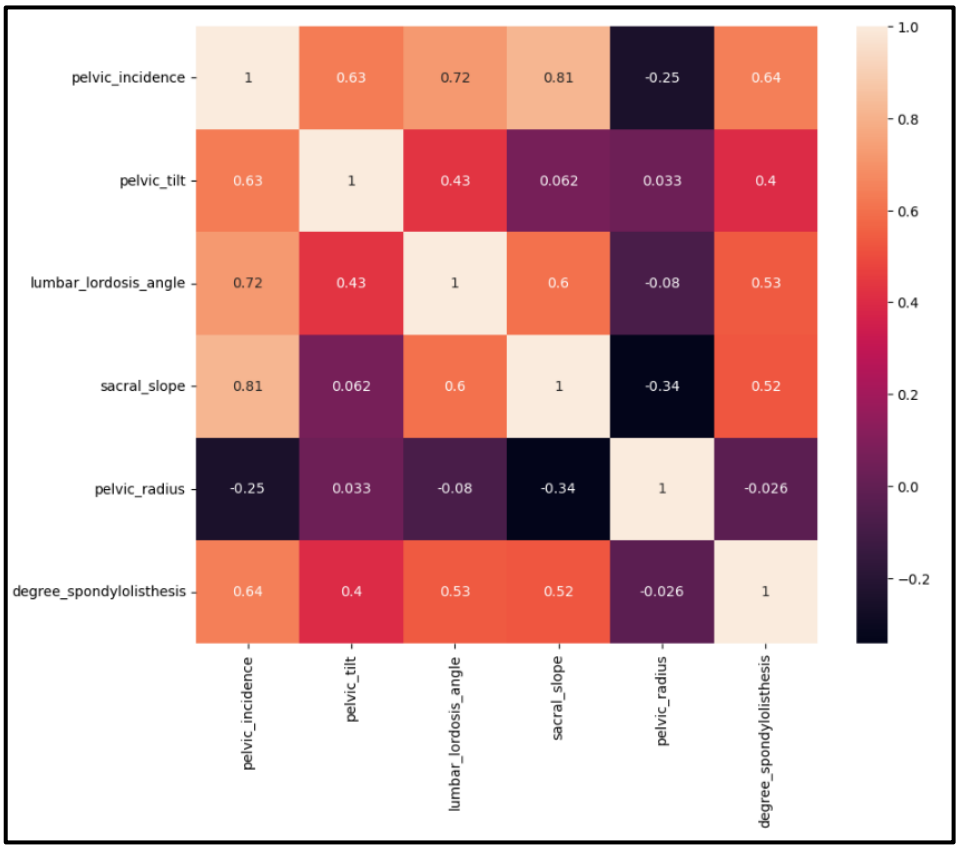
```
# Composiciónn de la cabecera
data[data.isnull().any(axis=1)].head(20)
```

pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class

Fuente: Elaboración propia

En la siguiente codificación, se presenta las columnas en el conjunto original y las columnas finales luego de haber realizado la revisión de datos faltantes. En la segunda línea de código se da conocer el porcentaje de columnas que no tengan datos faltantes, obteniendo un 100% ya que no existen datos que presenten missigno. En la tercera línea de código se da conocer la composición de la cabecera, las cuales no presentaron datos faltantes. Para continuar con el tratamiento de datos se generó la matriz de correlación de todas las variables. Donde se usó el método de Pearson para la generación de la gráfica.

Figura N° 76: Matriz de correlación de la data de entrenamiento

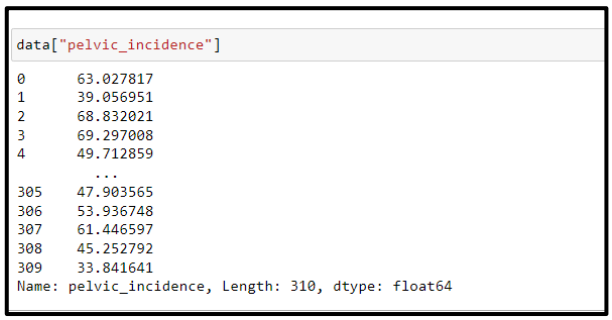


Fuente: Elaboración propia

En esta grafica se generó un mapa de calor en donde el valor 1 representa la correlación que tienen las variables y los distintos valores de correlación que se obtuvieron en función a las variables que se usaran en la fase de entrenamiento.

Para continuar con el tratamiento de datos, se generó las imputaciones para poder conocer a profundidad los datos con los que se trabajaron, realizando las imputaciones de cada variable.

Figura N° 77: Imputación de la variable pelvic_incidence

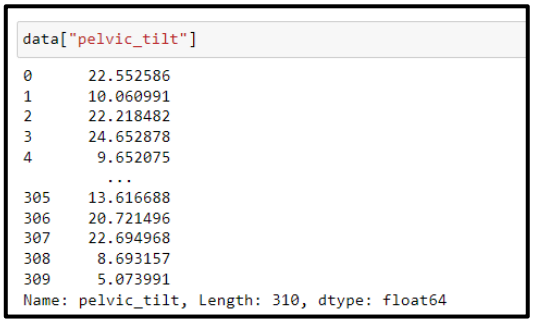


```
data["pelvic_incidence"]

0      63.027817
1      39.056951
2      68.832021
3      69.297008
4      49.712859
         ...
305    47.903565
306    53.936748
307    61.446597
308    45.252792
309    33.841641
Name: pelvic_incidence, Length: 310, dtype: float64
```

Fuente: Elaboración propia

Mediante ese código se da a conocer la cantidad de datos con los que trabajaran que son 310, el nombre de la variable a la cual se está realizando la imputación y el tipo de dato con de la variable, que es de tipo float64.

Figura N° 78: Imputación de la variable pelvic_tilt

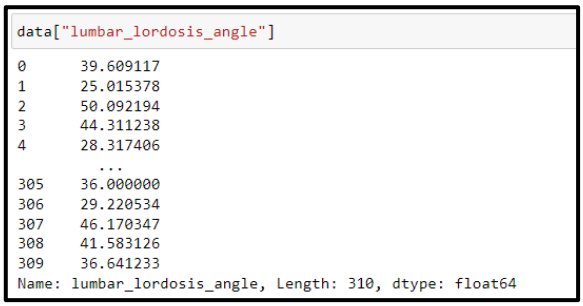


```
data["pelvic_tilt"]

0      22.552586
1      10.060991
2      22.218482
3      24.652878
4       9.652075
         ...
305    13.616688
306    20.721496
307    22.694968
308     8.693157
309     5.073991
Name: pelvic_tilt, Length: 310, dtype: float64
```

Fuente: Elaboración propia

Mediante ese código se da a conocer la cantidad de datos con los que trabajaran que son 310, el nombre de la variable a la cual se está realizando la imputación y el tipo de dato con de la variable, que es de tipo float64.

Figura N° 79: Imputación de la variable lumbar_lordosis_angle

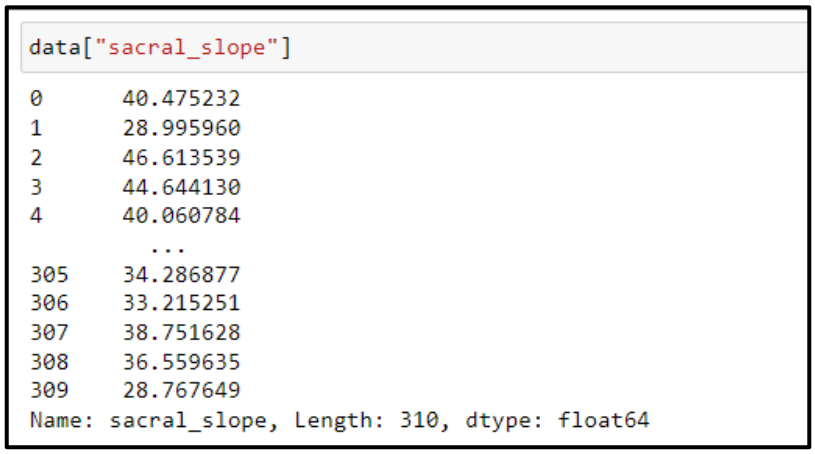


```
data["lumbar_lordosis_angle"]

0      39.609117
1      25.015378
2      50.092194
3      44.311238
4      28.317406
         ...
305    36.000000
306    29.220534
307    46.170347
308    41.583126
309    36.641233
Name: lumbar_lordosis_angle, Length: 310, dtype: float64
```

Fuente: Elaboración propia

Mediante ese código se da a conocer la cantidad de datos con los que trabajaran que son 310, el nombre de la variable a la cual se está realizando la imputación y el tipo de dato con de la variable, que es de tipo float64.

Figura N° 80: Imputación de la variable sacral_slope

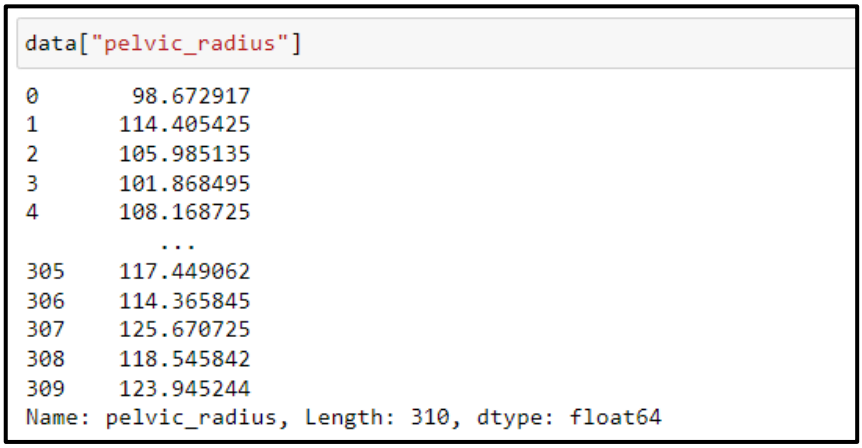


```
data["sacral_slope"]

0      40.475232
1      28.995960
2      46.613539
3      44.644130
4      40.060784
         ...
305    34.286877
306    33.215251
307    38.751628
308    36.559635
309    28.767649
Name: sacral_slope, Length: 310, dtype: float64
```

Fuente: Elaboración propia

Mediante ese código se da a conocer la cantidad de datos con los que trabajaran que son 310, el nombre de la variable a la cual se está realizando la imputación y el tipo de dato con de la variable, que es de tipo float64.

Figura N° 81: Imputación de la variable pelvic_radius



```
data["pelvic_radius"]

0       98.672917
1      114.405425
2      105.985135
3      101.868495
4      108.168725
          ...
305    117.449062
306    114.365845
307    125.670725
308    118.545842
309    123.945244
Name: pelvic_radius, Length: 310, dtype: float64
```

Fuente: Elaboración propia

Mediante ese código se da a conocer la cantidad de datos con los que trabajaran que son 310, el nombre de la variable a la cual se está realizando la imputación y el tipo de dato con de la variable, que es de tipo float64.

Figura N° 82: Imputación de la variable degree_spondylolisthesis

```
data["degree_spondylolisthesis"]

0      -0.254400
1       4.564259
2      -3.530317
3      11.211523
4       7.918501
         ...
305    -4.245395
306    -0.421010
307    -2.707880
308     0.214750
309    -0.199249
Name: degree_spondylolisthesis, Length: 310, dtype: float64
```

Fuente: Elaboración propia

Mediante ese código se da a conocer la cantidad de datos con los que trabajaran que son 310, el nombre de la variable a la cual se está realizando la imputación y el tipo de dato con de la variable, que es de tipo float64.

Figura N° 83 Imputación de la variable class

```
data["class"]

0      Hernia
1      Hernia
2      Hernia
3      Hernia
4      Hernia
        ...
305    Normal
306    Normal
307    Normal
308    Normal
309    Normal
Name: class, Length: 310, dtype: object
```

Fuente: Elaboración propia

Mediante ese código se da a conocer la cantidad de datos con los que trabajaran que son 310, el nombre de la variable a la cual se está realizando la imputación y el tipo de dato con de la variable, que es de tipo object

Continuando con el tratamiento de datos, se da conocer las columnas con las que cuenta la base de datos.

Figura N° 84: Columnas de la base de datos

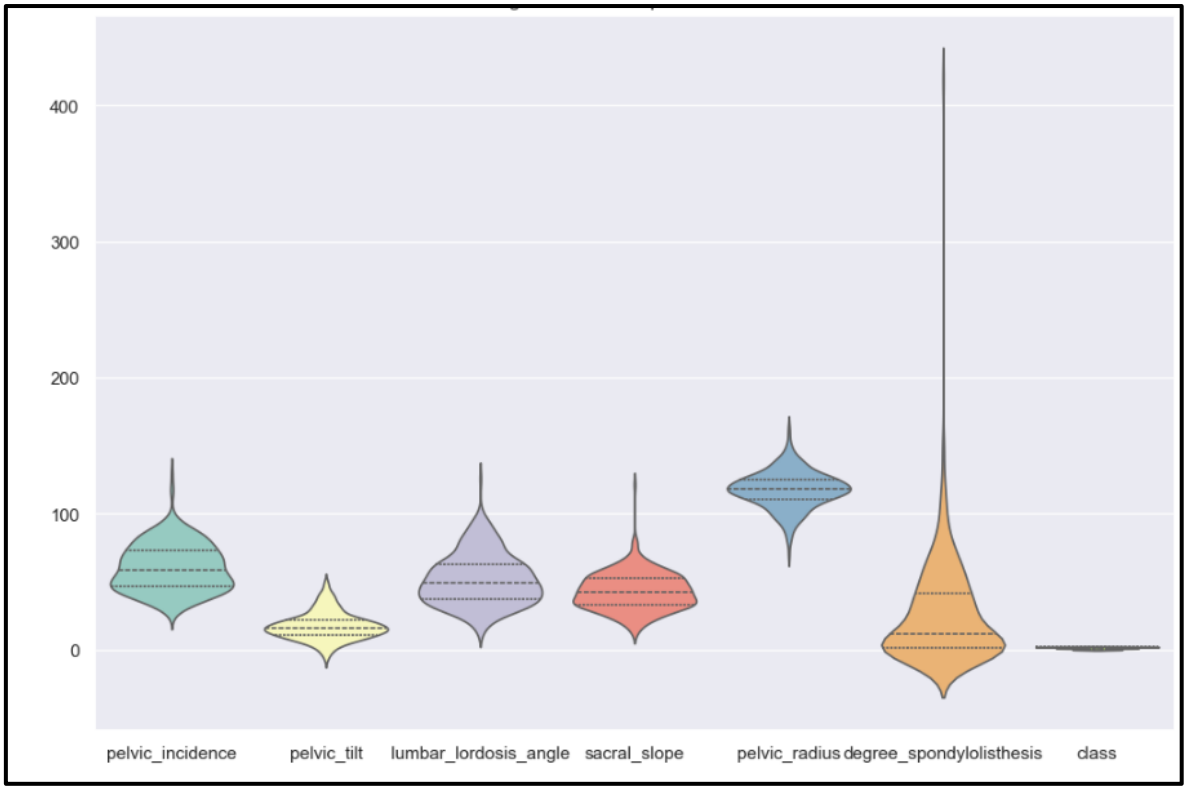
```
data.columns
```

```
Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class'],
      dtype='object')
```

Fuente: Elaboración propia

Continuando, se realizaron las siguientes gráficas para una mejor visualización de las variables.

Figura Nº 85: Diagrama de violín

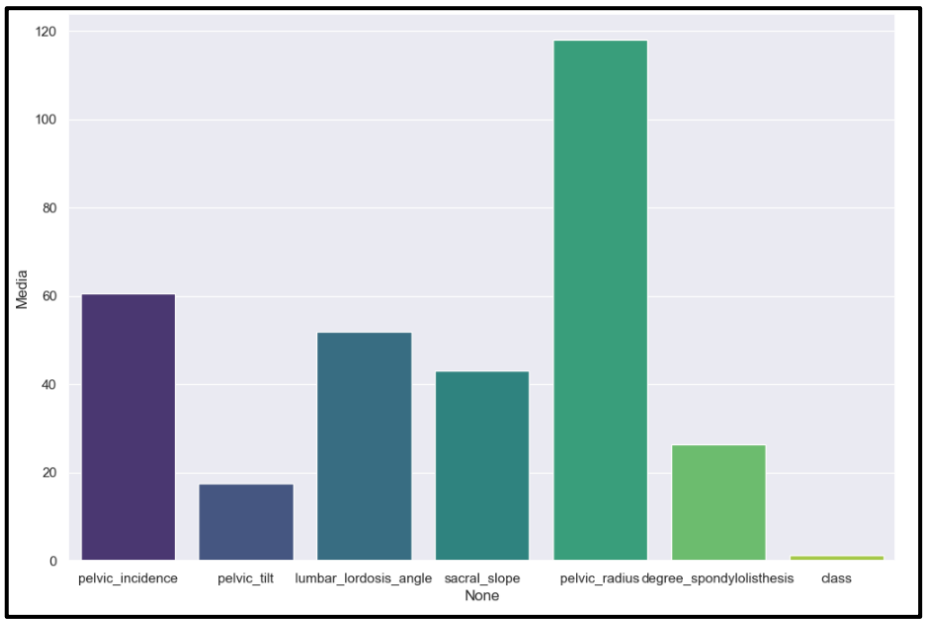


Fuente: Elaboración propia

Se realizo la gráfica de diagrama de violín para visualizar la dispersión de la distribución de los datos.

Se visualiza que la variable degree_spondylolisthesis tiene una mayor distribución en comparación con las otras variables. Por otro lado, la variable class es la que demuestra tener menos distribución.

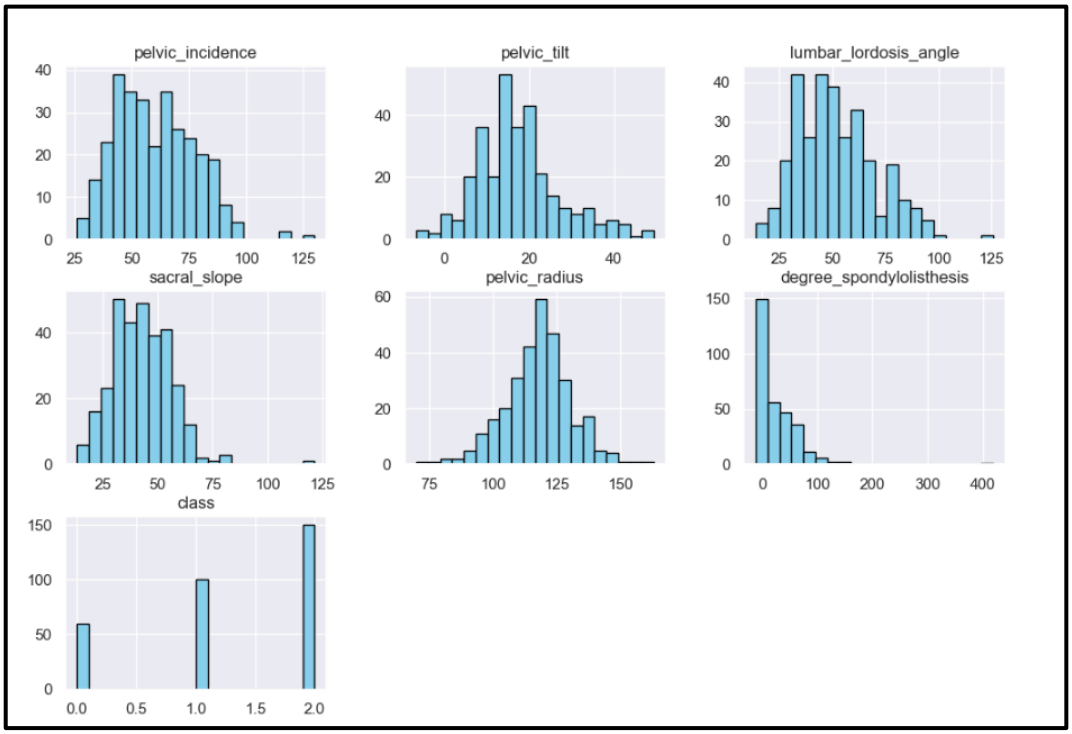
Figura Nº 86: Diagrama de barras



Fuente: Elaboración propia

Mediante esta grafica se puede visualizar la relación entre las variables categorías y numéricas. La barra de menor tamaño es class, ya que solo tiene 3 categorías.

Figura Nº 87: Histograma de variables



Fuente: Elaboración propia

Mediante el histograma, permitió visualizar el funcionamiento de cada variable, asi con la variable class, que sirve para la clasificación, tiene 3 barras que representan a la cantidad de categorías que existen en la variable class.

Luego de conocer las variables y realizar la limpieza de datos, se procedió a aplicar los métodos de selección de variables, en donde se tomará la variable class, como variable objetivo. La cual será representada en la siguiente tabla:

Tabla Nº 146: Selección de variables

Métodos	Variables
SelectKbest	pelvic_incidence, pelvic_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius, degree_spondylolisthesis
Puntuación RFE	pelvic_incidence, pelvic_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius, degree_spondylolisthesis
SELECTPERCENTILE	pelvic_incidence, lumbar_lordosis_angle, degree_spondylolisthesis

Fuente: Elaboración propia

Figura Nº 88: Resultados de selección de variables del método SelectKbest

```
Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis'],
      dtype='object')
```

Fuente: Elaboración propia

Figura Nº 89: Resultados de selección de variables del método RFE

```
Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis'],
     dtype='object')
```

Fuente: Elaboración propia

Figura Nº 90: Resultados de selección de variables del método SelectPercentile

```
Index(['pelvic_incidence', 'lumbar_lordosis_angle',
       'degree_spondylolisthesis'],
      dtype='object')
```

Fuente: Elaboración propia

Se visualiza que los métodos SelectKbest y puntuación RFE mostraron resultados iguales. En base a ello se utilizaron los variables: pelvic_incidence, pelvic_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius y degree_spondylolisthesis.

Luego de realizar la selección de variables a aplicar el método de balanceo de datos o resampling. Se aplicaron los métodos Undersampling y Oversampling.

En la aplicación del método Undersampling se utilizó los métodos Random Undersampling y Tomek-link, se obtuvo los siguientes resultados:

Figura Nº 91: Resultados de Random Undersampling

```
Conteo de clases antes de aplicar Random Undersampling: Counter({'Spondylolisthesis': 150, 'Normal': 100, 'Hernia': 60})
Conteo de clases después de aplicar Random Undersampling: Counter({'Hernia': 60, 'Normal': 60, 'Spondylolisthesis': 60})
```

Fuente: Elaboración propia

Se obtuvo como resultados que mediante la aplicación del método random undersampling, el número de clases de instancias se ha disminuido a 60 para igualar el número de instancias de la clase Hernia.

Figura Nº 92: Resultados de Tomek Link

```
Conteo de clases antes de aplicar Tomek Links: Counter({'Spondylolisthesis': 150, 'Normal': 100, 'Hernia': 60})
Conteo de clases después de aplicar Tomek Links: Counter({'Spondylolisthesis': 146, 'Normal': 91, 'Hernia': 60})
```

Fuente: Elaboración propia

Se visualiza que que el número de instancias en las clases mayoritarias ('Spondylolisthesis' y 'Normal') se ha reducido, lo que puede indicar que se eliminaron algunas instancias de esas clases para mejorar la separación con respecto a la clase minoritaria ('Hernia').

En la aplicación del método Oversampling se utilizó los métodos Random Oversampling y SMOTE, se obtuvo los siguientes resultados:

Figura Nº 93: Resultados de Random Undersampling

```
Conteo de clases antes de aplicar Random Oversampling: Counter({'Spondylolisthesis': 150, 'Normal': 100, 'Hernia': 60})
Conteo de clases después de aplicar Random Oversampling: Counter({'Hernia': 150, 'Spondylolisthesis': 150, 'Normal': 150})
```

Fuente: Elaboración propia

En este resultado específico, el número de instancias de cada clase se ha aumentado a 150 para igualar la cantidad de instancias en la clase con más observaciones.

Figura Nº 94: Resultados de SMOTE

```
Conteo de clases antes de aplicar SMOTE: Counter({'Spondylolisthesis': 150, 'Normal': 100, 'Hernia': 60})
Conteo de clases después de aplicar SMOTE: Counter({'Hernia': 150, 'Spondylolisthesis': 150, 'Normal': 150})
```

Fuente: Elaboración propia

En este resultado específico, se han generado instancias sintéticas de 'Hernia' para aumentar su número a 150, igualando así la cantidad de instancias en las clases mayoritarias.

Los algoritmos de Random undersampling y SMOTE tuvieron los mismos resultados, por otro lado, el método tomek link obtuvo valores similares a los métodos de undersampling.

Cuarta Etapa: Minería de datos

Para identificar las relaciones y el reconocimiento de patrones de datos del modelo rendimiento académico con las variables seleccionadas se aplicó en 10 algoritmos de aprendizaje automático para realizar el entrenamiento y la validación en el entorno interactivo Jupyter.

Para los 12 algoritmos y 5 modelos stacking se realizó la importación en general de las siguientes librerías y algoritmos específicos.

Conexión con los datos e importación de librerías:

Figura Nº 95: Conexión de la data

```
In [2]: data = pd.read_csv('column_3C.csv')
        data.head()
```

Out[2]:

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254400	Hernia
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	Hernia
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	Hernia
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	Hernia
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	Hernia

Fuente: Elaboración propia

Importación de librerías:

Figura Nº 96: Librerías Generales

```
# Librerías y gráficos
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.metrics import confusion_matrix

# Selección de mejores hiperparámetros
from sklearn.model_selection import train_test_split, GridSearchCV
# Seleccion de mejores hiperparámetros para el algoritmo Redes Neuronales Convolucionales
from kerastuner.tuners import RandomSearch
```

Fuente: Elaboración propia

Pandas (import pandas as pd):

Pandas es una poderosa biblioteca de manipulación de datos para Python. Proporciona estructuras de datos como DataFrame para el manejo y análisis eficiente de datos.

Matplotlib (import matplotlib.pyplot as plt):

Matplotlib es una biblioteca ampliamente utilizada para graficar en Python. El módulo pyplot proporciona una interfaz similar a MATLAB para crear visualizaciones estáticas, animadas e interactivas.

Seaborn (import seaborn as sns):

Seaborn se construye sobre Matplotlib y proporciona una interfaz de alto nivel para la visualización estadística de datos. Simplifica el proceso de creación de gráficos estadísticos informativos y atractivos.

TensorFlow (import tensorflow as tf):

TensorFlow es una biblioteca de aprendizaje automático de código abierto desarrollada por el equipo de Google Brain. Se utiliza ampliamente para construir y entrenar diversos modelos de aprendizaje automático, incluidas las redes neuronales.

Scikit-Learn (from sklearn.metrics import confusion_matrix, from sklearn.model_selection import train_test_split, GridSearchCV):

Scikit-Learn es una biblioteca de aprendizaje automático que proporciona herramientas simples y eficientes para la minería de datos y el análisis de datos. Incluye diversas herramientas para clasificación, regresión, agrupación, entre otras. Aquí está el desglose:

train_test_split: Esta función se utiliza para dividir conjuntos de datos en conjuntos de entrenamiento y prueba, lo cual es crucial para la evaluación del modelo.

GridSearchCV: Esta clase realiza una búsqueda exhaustiva sobre una cuadrícula de parámetros especificada, optimizando hiperparámetros para un estimador dado (modelo).

KerasTuner (from kerastuner.tuners import RandomSearch):

Propósito: KerasTuner es una biblioteca para la optimización de hiperparámetros de modelos Keras. En este caso, estás utilizando RandomSearch de KerasTuner para realizar una búsqueda aleatoria en el espacio de hiperparámetros para optimizar una Red Neuronal Convolucional (CNN).

Figura Nº 97: Librerías para los algoritmos

```
# Librerías de algoritmos
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import NearestCentroid
from sklearn.ensemble import RandomForestClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
```

Fuente: Elaboración propia

Decision Tree (DecisionTreeClassifier):

Propósito: Se utiliza para construir modelos de árboles de decisión, que son estructuras de árbol que representan decisiones y sus posibles consecuencias.

Gradient Boosting (GradientBoostingClassifier):

Propósito: Implementa el algoritmo de Boosting, específicamente el método de Gradient Boosting, que combina varios modelos débiles para formar un modelo más robusto y preciso.

K-Nearest Neighbors (KNN) (KNeighborsClassifier):

Propósito: Es un algoritmo de clasificación que asigna una etiqueta a un punto de datos basándose en las etiquetas de los puntos vecinos más cercanos en el espacio de características.

Naive Bayes (GaussianNB):

Propósito: Implementa el clasificador Naive Bayes, un algoritmo de aprendizaje supervisado basado en el teorema de Bayes, que asume independencia condicional entre las características.

Nearest Centroid (NearestCentroid):

Propósito: Es un clasificador que asigna un punto de datos a la clase cuyo centroide es el más cercano.

Random Forest (RandomForestClassifier):

Propósito: Es un conjunto de árboles de decisión, donde cada árbol vota por una clase, y la clase con más votos se elige como la predicción final.

Para los algoritmos de redes neuronales y redes neuronales convolucionales, se usaron las librerías Sequential y Dense

Red Neuronal Convolucional (CNN):

Propósito: Se utiliza para construir redes neuronales convolucionales, especialmente eficientes para tareas de visión por computadora al procesar datos en forma de mallas.

Redes Neuronales (Sequential, Dense):

Propósito: Permite la construcción de redes neuronales artificiales. Sequential es un modelo lineal donde puedes apilar capas, y Dense define capas completamente conectadas.

Regresión Logística (LogisticRegression):

Propósito: Implementa el algoritmo de regresión logística, que se utiliza para problemas de clasificación binaria y multiclase.

Support Vector Machine (SVM) (SVC):

Propósito: Implementa el algoritmo de máquinas de soporte vectorial, que es eficaz tanto para problemas de clasificación como de regresión.

XGBoost (XGBClassifier):

Propósito: Es una implementación eficiente de Gradient Boosting que destaca por su velocidad y rendimiento.

AdaBoost (AdaBoostClassifier):

Propósito: Implementa el algoritmo AdaBoost, que combina varios modelos débiles para mejorar el rendimiento del modelo final.

Figura Nº 98: Librerías para las métricas

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import matthews_corrcoef
```

Fuente: Elaboración propia

Accuracy(exactitud): from sklearn.metrics import accuracy_score

Mide la proporción de predicciones correctas en relación con el total de predicciones. Es adecuada para conjuntos de datos balanceados, pero puede ser engañosa en conjuntos desbalanceados.

Precision (Precisión): from sklearn.metrics import precision_score

Mide la proporción de instancias positivas correctamente clasificadas entre todas las instancias clasificadas como positivas. Es útil cuando el coste de los falsos positivos es alto.

Recall (Sensibilidad): from sklearn.metrics import recall_score

Mide la proporción de instancias positivas correctamente clasificadas entre todas las instancias que son realmente positivas. Es útil cuando el coste de los falsos negativos es alto.

F1-Score: from sklearn.metrics import f1_score

Es una medida que combina precisión y recall en un solo número. Es útil cuando se desea equilibrar ambas métricas.

AUC (Área bajo la curva ROC): from sklearn.metrics import roc_auc_score

Mide la capacidad del modelo para discriminar entre clases positivas y negativas. Cuanto mayor sea el AUC, mejor será el modelo para clasificar las instancias.

MCC (Coeficiente de correlación de Matthews): from sklearn.metrics import matthews_corrcoef

Es una medida que tiene en cuenta los cuatro valores de la matriz de confusión. Proporciona una visión equilibrada del rendimiento del modelo.

- **Particionamiento de la data**

Figura Nº 99: Particionamiento de la data

```
X = data[['pelvic_incidence','pelvic_tilt','lumbar_lordosis_angle','sacral_slope',
          'pelvic_radius','degree_spondylolisthesis']].values

Y = data['class']
Y.shape
```

Fuente: Elaboración propia

Selección de las variables

Contiene las 6 variables establecidas para el modelo.

Definición de las variables

Se asigna a la variable “X” y representa la matriz de características.

Se asigna a la variable “Y” y representa la variable objetivo.

División del conjunto de entrenamiento y validación

Se dividen los datos en un 80% y 20% para el conjunto de entrenamiento y validación, respectivamente

Continuando, se procedió a generar la búsqueda de hiperparámetros para cda algoritmo.

- **Decision Tree**

Figura Nº 100: Hiperparámetros - DT

```
# Definir un rango de hiperparámetros para la búsqueda de cuadrícula
param_grid = {
    'criterion': ['gini', 'entropy'],  # Criterio de división
    'max_depth': [None, 10, 20, 30],  # Profundidad máxima del árbol
    'min_samples_split': [2, 5, 10],  # Número mínimo de muestras requeridas para dividir un nodo
    'min_samples_leaf': [1, 2, 4]  # Número mínimo de muestras requeridas en un nodo hoja
}
```

```
# Crear un clasificador de árbol de decisiones
tree_classifier = DecisionTreeClassifier(random_state=42)
```

```
# Realizar la búsqueda de cuadrícula con validación cruzada
grid_search = GridSearchCV(tree_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Obtener los mejores hiperparámetros encontrados
best_params = grid_search.best_params_
print("Mejores hiperparámetros:", best_params)
```

```
Mejores hiperparámetros: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 2}
```

```
# Entrenar el modelo de árbol de decisiones con los mejores hiperparámetros en los datos de entrenamiento
best_tree = DecisionTreeClassifier(random_state=42, **best_params)
best_tree.fit(X_train, y_train)
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió un rango de hiperparámetros para la busqueda de cuadricula, con los parámetros criterion, max_depth, min_samples_split y min_samples_leaf. Continuando, se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mismos valores que se usaron en la búsqueda de cuadrícula, pero se agregó el parámetro entropy.

Evaluación de las métricas

Figura Nº 101: Evaluación de las métricas - DT

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Calcular y mostrar la precisión
accuracy = accuracy_score(y_test, y_pred)
print(f'Acuraccy del modelo: {accuracy}')

# Almacenar precisión
algorithm_1_accuracy = accuracy
```

```
Acuraccy del modelo: 0.8225806451612904
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_1_precision = precision
```

```
Precisión del modelo de stacking: 0.7757352941176471
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred, average='macro')
print("F1-score del modelo:", f1)

# Almacenar precisión
algorithm_1_f1 = f1
```

```
F1-score del modelo: 0.7799028536733456
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, y_pred)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_1_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.7237461832398165
```

Fuente: Elaboración propia

En las figuras Nº 101, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score y MCC, en donde se visualiza el código y el rendimiento de cada métrica. También se almacenó cada métrica para poder realizar las gráficas y así poder comparar el rendimiento de los modelos.

Figura Nº 101: Evaluación de las métricas - DT

```
# Recall
recall = recall_score(y_test, y_pred, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_1_recall = recall
```

```
Recall: 0.8028807814992026
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_1_spe = specificity
```

```
Specificity for class 0: 0.86
Specificity for class 1: 0.90
Specificity for class 2: 1.00
Average Specificity: 0.92
```

Fuente: Elaboración propia

Figura Nº 101 Evaluación de las métricas - DT

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize

# Crear y entrenar el modelo de árbol de decisiones
tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred_proba = tree_model.predict_proba(X_test)

# Binarizar las etiquetas del conjunto de prueba
y_true_binary = label_binarize(y_test, classes=tree_model.classes_)

# Calcular el AUC usando la estrategia micro-average
auc_micro = roc_auc_score(y_true_binary, y_pred_proba, average='micro')

# Calcular el AUC usando la estrategia macro-average
auc_macro = roc_auc_score(y_true_binary, y_pred_proba, average='macro')

# Imprimir el AUC para micro-average y macro-average
print(f'AUC (Micro-average): {auc_micro:.2f}')
print(f'AUC (Macro-average): {auc_macro:.2f}')

# Almacenar precisión
algorithm_1_auc = auc_macro
```

```
AUC (Micro-average): 0.85
AUC (Macro-average): 0.84
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **Gradient Boosting**

Figura Nº 102 Hiperparámetros - GB

```
# Define los hiperparámetros que deseas ajustar
param_grid = {
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 4, 8],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 0.9, 1.0],
    'max_features': [None, 'sqrt', 'log2']
}
```

```
# Crea el modelo de Gradient Boosting
model = GradientBoostingClassifier(random_state=42)
```

```
from sklearn.model_selection import RandomizedSearchCV

random_search = RandomizedSearchCV(model, param_distributions=param_grid, n_iter=50, cv=3, scoring='accuracy', n_jobs=-1)
random_search.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=42),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1, 0.2],
                                        'max_depth': [3, 5, 7, 9],
                                        'max_features': [None, 'sqrt', 'log2'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 4, 8],
                                        'n_estimators': [50, 100, 150, 200],
                                        'subsample': [0.8, 0.9, 1.0]},
                   scoring='accuracy')
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

```
# Imprime los mejores hiperparámetros
print("Mejores hiperparámetros:")
print(random_search.best_params_)
```

```
Mejores hiperparámetros:
{'subsample': 0.9, 'n_estimators': 50, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 5,
'learning_rate': 0.05}
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió un rango de hiperparámetros para la búsqueda de cuadricula, con los parámetros learning_rate, n_estimators, max_depth, min_samples_split, min_samples_leaf, subsample y max_features. Continuando, se creó el modelo de Gradient Boosting y finalmente se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mismos valores que se usaron en la

búsqueda de cuadrícula, pero se obtuvo valores precisos de cada parámetro, que ayudará a tener un mejor rendimiento del algoritmo.

Evaluación de las métricas

Figura Nº 103: Evaluación de las métricas - GB

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score

accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo optimizado: {accuracy}')

# Almacenar precisión
algorithm_2_accuracy = accuracy
```

```
Precisión del modelo optimizado: 0.7741935483870968
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_2_precision = precision
```

```
Precisión del modelo de stacking: 0.677807486631016
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred, average='macro')
print("F1-score del modelo de stacking:", f1)

# Almacenar precisión
algorithm_2_f1 = f1
```

```
F1-score del modelo de stacking: 0.6766178266178265
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, y_pred)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_2_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.6290245103533908
```

```
# Recall
recall = recall_score(y_test, y_pred, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_2_recall = recall
```

```
Recall: 0.6759259259259259
```

Fuente: Elaboración propia

En la figura Nº 103, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC y recall, en donde se visualiza el código y el rendimiento de cada métrica. También se almacenó cada métrica para poder realizar las gráficas y así poder comparar el rendimiento de los modelos.

Figura Nº 103: Evaluación de las métricas - GB

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_2_spe = specificity
```

```
Specificity for class 0: 0.86
Specificity for class 1: 0.86
Specificity for class 2: 0.94
Average Specificity: 0.89
```

```
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Realiza predicciones de probabilidades en el conjunto de prueba
y_prob = best_model.predict_proba(X_test)

# Calcular el AUC-ROC
auc_roc = roc_auc_score(y_test, y_prob, multi_class='ovr')
print(f'AUC-ROC del modelo optimizado: {auc_roc}')

algorithm_2_auc = auc_roc
```

```
AUC-ROC del modelo optimizado: 0.9337373737373738
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, multi_class=”ovr” sirvió para las 3 clases, ya que el problema se descompone en múltiples subproblemas de clasificación binaria.

- **K-Nearest Neighbort**

Figura Nº 104: Hiperparámetros - KNN

```
# Normalizar los datos para mejorar el rendimiento del modelo
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Definir el modelo KNN
knn = KNeighborsClassifier()

# Definir la cuadrícula de hiperparámetros a explorar
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [20, 30, 40]
}

# Realizar la búsqueda en cuadrícula
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)
```

GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier

```
# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_

# Imprimir los mejores hiperparámetros
print("Mejores hiperparámetros encontrados:", best_params)
```

```
Mejores hiperparámetros encontrados: {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 7, 'weights': 'distance'}
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió un rango de hiperparámetros para la busqueda de cuadricula, con los parámetros n_neighbors, weigths, algorithm, leaf_size. Continuando, se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mismos valores que se usaron en la búsqueda de cuadrícula, pero se agregó el parámetro auto.

Figura Nº 105: Evaluación de rendimiento del modelo - KNN

```
# Obtener el modelo con los mejores hiperparámetros
best_knn = grid_search.best_estimator_


# Hacer predicciones en el conjunto de prueba
y_pred = best_knn.predict(X_test_scaled)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print('Classification Report:\n', report)
```

```
Accuracy: 0.8387096774193549
```

Fuente: Elaboración propia

En estas celdas de código se obtuvo los mejores hiperparámetros del modelo KNN, se realizaron las predicciones usaron los mejores hiperparámetros y se evaluó el rendimiento del modelo, el cual obtuvo un 0.837 de Acuraccy.

Figura Nº 106: Optimización del algoritmo - KNN

```
# Definir la cuadrícula de hiperparámetros a explorar
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [20, 30, 40],
    'p': [1, 2]
}

# Realizar la búsqueda en cuadrícula
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)
```

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

```
# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_


# Imprimir los mejores hiperparámetros
print("Mejores hiperparámetros encontrados:", best_params)
```

```
Mejores hiperparámetros encontrados: {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 9, 'p': 2, 'weights': 'distance'}
```

```
# Obtener el modelo con los mejores hiperparámetros
best_knn = grid_search.best_estimator_

# Hacer predicciones en el conjunto de prueba
y_pred = best_knn.predict(X_test_scaled)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print('Classification Report:\n', report)
```

```
Accuracy: 0.8548387096774194
```

Fuente: Elaboración propia

En estas celdas de código se agregó el hiperparámetro p, cuando p = 1, se utiliza la distancia de Manhattan y cuando p = 2, se utiliza la distancia euclidiana. Se evaluó el modelo con los nuevos hiperparámetros y se obtuvo un Acuraccy de 0.854.

Evaluación de las métricas

Figura Nº 107: Evaluación de las métricas - KNN

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Calcular y mostrar la precisión
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo: {accuracy}')

# Almacenar precisión
algorithm_3_accuracy = accuracy
```

```
Precisión del modelo: 0.8548387096774194
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_3_precision = precision
```

```
Precisión del modelo de stacking: 0.8161493477282952
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_3_f1 = f1
```

```
Precisión del modelo de stacking: 0.8249858517260895
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, y_pred)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_3_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.7669331079306455
```

```
# Recall
recall = recall_score(y_test, y_pred, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_3_recall = recall
```

```
Recall: 0.8379685007974481
```

Fuente: Elaboración propia

En la figura Nº 107, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC y recall, en donde se visualiza el código y el rendimiento de cada métrica. También se almacenó cada métrica para poder realizar las gráficas y así poder comparar el rendimiento de los modelos.

Figura Nº 107: Evaluación de las métricas - KNN

```
from sklearn.preprocessing import label_binarize
# Crear y entrenar el modelo KNN
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred_proba = knn_model.predict_proba(X_test)

# Calcular el AUC usando la estrategia micro-average
y_true_binary = label_binarize(y_test, classes=knn_model.classes_)
auc_micro = roc_auc_score(y_true_binary, y_pred_proba, average='micro')

# Calcular el AUC usando la estrategia macro-average
auc_macro = roc_auc_score(y_true_binary, y_pred_proba, average='macro')

# Imprimir el AUC para micro-average y macro-average
print(f'AUC (Micro-average): {auc_micro:.2f}')
print(f'AUC (Macro-average): {auc_macro:.2f}')

# Almacenar precisión
algorithm_3_auc = auc_macro
```

```
AUC (Micro-average): 0.97
AUC (Macro-average): 0.95
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_3_spe = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.90
Specificity for class 2: 0.96
Average Specificity: 0.93
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **Naive Bayes**

Figura Nº 108: Hiperparámetros - NB

```
# Definir los hiperparámetros que deseas probar
parametros = {}

# Crear el modelo Naive Bayes Gaussiano
modelo_naive_bayes = GaussianNB()

# Configurar la búsqueda en cuadrícula
busqueda_en_cuadricula = GridSearchCV(modelo_naive_bayes, parametros, cv=5, scoring='accuracy')

# Realizar la búsqueda en cuadrícula en los datos de entrenamiento
busqueda_en_cuadricula.fit(X_train, y_train)
```

GridSearchCV
estimator: GaussianNB
GaussianNB

```
# Obtener el mejor modelo con los mejores hiperparámetros
mejor_modelo = busqueda_en_cuadricula.best_estimator_

# Realizar predicciones en el conjunto de prueba con el mejor modelo
predicciones = mejor_modelo.predict(X_test)
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió la variable parámetros, para guardar los parámetros. Se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 109: Evaluación de las métricas - NB

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score

# Calcular y mostrar la precisión
accuracy = accuracy_score(y_test, predicciones)
print(f'Precisión del modelo: {accuracy}')

# Almacenar precisión
algorithm_4_accuracy = accuracy
```

```
Precisión del modelo: 0.8709677419354839
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, predicciones, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_4_precision = precision
```

```
Precisión del modelo de stacking: 0.8333836098541981
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, predicciones, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_4_f1 = f1
```

```
Precisión del modelo de stacking: 0.8258585858585858
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, predicciones)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_4_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.7895578683249794
```

```
# Recall
recall = recall_score(y_test, predicciones, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_4_recall = recall
```

```
Recall: 0.8240740740740741
```

Fuente: Elaboración propia

En la figura Nº 109, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC y recall, en donde se visualiza el código y el rendimiento de cada métrica. También se almacenó cada métrica para poder realizar las gráficas y así poder comparar el rendimiento de los modelos.

Figura Nº 109: Evaluación de las métricas - NB

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_4_spe = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.95
Specificity for class 2: 0.92
Average Specificity: 0.93
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np


# Crear y entrenar el modelo Naive Bayes (GaussianNB)
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred_proba = nb_model.predict_proba(X_test)

# Binarizar las etiquetas
y_true_binary = label_binarize(y_test, classes=nb_model.classes_)
auc_micro = roc_auc_score(y_true_binary, y_pred_proba, average='micro')

# Calcular el AUC usando la estrategia macro-average
auc_macro = roc_auc_score(y_true_binary, y_pred_proba, average='macro')

# Imprimir el AUC para micro-average y macro-average
print(f'AUC (Micro-average): {auc_micro:.2f}')
print(f'AUC (Macro-average): {auc_macro:.2f}')

# Almacenar precisión
algorithm_4_auc = auc_macro
```

```
AUC (Micro-average): 0.96
AUC (Macro-average): 0.95
```

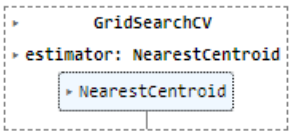
Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **Nearest Centroid**

Figura Nº 110: Hiperparámetros - NC

```
# Definir el espacio de búsqueda de hiperparámetros
param_grid = {
    'shrink_threshold': [None, 0.1, 0.5, 1.0],  # Ejemplos de valores para el hiperparámetro shrink_threshold
}

from sklearn.model_selection import GridSearchCV
# Configurar la búsqueda grid con validación cruzada
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')

# Realizar la búsqueda grid en los datos de entrenamiento
grid_search.fit(X_train, y_train)
```




```
# Obtener el mejor modelo y sus hiperparámetros
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# Realizar predicciones en el conjunto de prueba con el mejor modelo
predictions = best_model.predict(X_test)

print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'shrink_threshold': None}
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió la variable parámetros, para guardar los parámetros. Se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 111: Evaluación de las métricas - NC

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Calcular y mostrar la precisión
accuracy = accuracy_score(y_test, predictions)
print(f'Precisión del modelo: {accuracy}')

# Almacenar precisión
algorithm_5_accuracy = accuracy
```

```
Precisión del modelo: 0.8064516129032258
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, predictions, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_5_precision = precision
```

```
Precisión del modelo de stacking: 0.7716149347728295
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, predictions, average='macro')
print("F1-score del modelo de stacking:", f1)

# Almacenar precisión
algorithm_5_f1 = f1
```

```
F1-score del modelo de stacking: 0.7634408602150539
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, predictions)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_5_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.7042140165013202
```

```
# Recall
recall = recall_score(y_test, predictions, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_5_recall = recall
```

```
Recall: 0.7928240740740741
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_5_spe = specificity
```

```
Specificity for class 0: 0.83
Specificity for class 1: 0.93
Specificity for class 2: 0.95
Average Specificity: 0.90
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **Random Forest**

Figura Nº 112: Hiperparámetros - RF

```
param_grid = {
    'n_estimators': [100, 200, 300],      # Número de árboles en el bosque
    'max_depth': [None, 10, 20, 30],     # Profundidad máxima de los árboles
    'min_samples_split': [2, 5, 10],    # Número mínimo de muestras requeridas para dividir un nodo
    'min_samples_leaf': [1, 2, 4]       # Número mínimo de muestras requeridas en un nodo hoja
}
```

```
# Crear el clasificador Random Forest
rf = RandomForestClassifier(random_state=42)
```

```
# Crear un objeto GridSearchCV para encontrar los mejores hiperparámetros
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
```

```
# Ajustar el modelo a los datos de entrenamiento
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_
```

```
# Crear un nuevo clasificador Random Forest con los mejores hiperparámetros
best_rf = RandomForestClassifier(random_state=42, **best_params)
```

```
# Entrenar el modelo final en los datos de entrenamiento
best_rf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=300, random_state=42)
```

```
# Realizar predicciones en el conjunto de prueba
y_pred = best_rf.predict(X_test)
```

```
# Calcular la precisión del modelo en el conjunto de prueba
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Mejores hiperparámetros:", best_params)
print("Precisión del modelo optimizado:", accuracy)
```

```
Mejores hiperparámetros: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Precisión del modelo optimizado: 0.9032258064516129
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió la variable parámetros, para guardar los parámetros. Se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 113 Evaluación de las métricas - RF

```
# Exactitud (Accuracy)
accuracy = accuracy_score(y_test, y_pred_rf)
print(f'Exactitud (Accuracy): {accuracy:.4f}')

# Almacenar precisión
algorithm_6_accuracy = accuracy
```

```
Exactitud (Accuracy): 0.9032
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred_rf, average='macro')
print(f'Precisión (Precision): {precision:.4f}')

# Almacenar precisión
algorithm_6_precision = precision
```

```
Precisión (Precision): 0.8604
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred_rf, average='macro')
print(f'Puntuación F1 (F1-score): {f1:.4f}')

# Almacenar precisión
algorithm_6_f1 = f1
```

```
Puntuación F1 (F1-score): 0.8663
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, y_pred_rf)
print(f'Coeficiente de correlación de Matthews (MCC): {mcc:.4f}')

# Almacenar precisión
algorithm_6_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.8440
```

```
# Recall
recall = recall_score(y_test, y_pred_rf, average='macro')
print(f'Recall: {recall:.4f}')

# Almacenar precisión
algorithm_6_recall = recall
```

```
Recall: 0.8763
```

Fuente: Elaboración propia

Figura Nº 113 Evaluación de las métricas - RF

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_6_spe = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.95
Specificity for class 2: 1.00
Average Specificity: 0.96
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize

# Crear y entrenar el modelo Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Realizar predicciones de probabilidad en el conjunto de prueba
y_pred_proba = rf_model.predict_proba(X_test)

# Binarizar las etiquetas
y_true_binary = label_binarize(y_test, classes=rf_model.classes_)

# Calcular el AUC usando la estrategia micro-average
auc_micro = roc_auc_score(y_true_binary.ravel(), y_pred_proba.ravel(), average='micro')

# Calcular el AUC usando la estrategia macro-average
auc_macro = roc_auc_score(y_true_binary, y_pred_proba, average='macro')

# Imprimir el AUC para micro-average y macro-average
print(f'AUC (Micro-average): {auc_micro:.2f}')
print(f'AUC (Macro-average): {auc_macro:.2f}')

# Almacenar precisión
algorithm_6_auc = auc_macro
```

```
AUC (Micro-average): 0.98
AUC (Macro-average): 0.97
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **Redes Neuronales convolucionales**

Figura Nº 114: Hiperparámetros - RNC

```
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Inicializar el sintonizador
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=5,  # Número de combinaciones de hiperparámetros a probar
    directory='tuner_directory',  # Directorio para almacenar resultados
    project_name='my_tuning_project'
)

# Realizar la búsqueda de hiperparámetros
tuner.search(X_train, y_train_encoded, epochs=10, validation_split=0.2)

# Obtener los mejores hiperparámetros
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"\nMejores hiperparámetros: {best_hps}")
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió la variable parámetros, para guardar los parámetros. Se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 115: Evaluación de las métricas - RNC

```
# Calcular métricas adicionales
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
roc_auc = roc_auc_score(pd.get_dummies(y_test), y_pred_probs, multi_class='ovr')
conf_mat = confusion_matrix(y_test, y_pred)

# Mostrar resultados
print(f'Acuraccy del modelo de red neuronal: {accuracy:.2f}')
print(f'Precisión: {precision:.2f}')
print(f'Recall ponderado: {recall:.2f}')
print(f'Puntuación F1 ponderada: {f1:.2f}')
print(f'AUC: {roc_auc:.2f}')
print('Matriz de Confusión:')
print(conf_mat)


# Almacenar precisión
algorithm_8_accuracy = accuracy
algorithm_8_precision = precision
algorithm_8_recall = recall
algorithm_8_f1 = f1
algorithm_8_auc = roc_auc
```

Fuente: Elaboración propia

Figura Nº 115: Evaluación de las métricas - RNC

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_8_spe = specificity
```

```
Specificity for class 0: 1.00
Specificity for class 1: 0.79
Specificity for class 2: 0.86
Average Specificity: 0.88
```

```
# Calcular MCC
mcc = matthews_corrcoef(y_test, y_pred)
print(f'MCC: {mcc:.2f}')

# Almacenar precisión
algorithm_8_mcc = mcc
```

```
MCC: 0.70
```

```
# Pérdida final del modelo
final_loss = history.history['loss'][-1]
print(f'Pérdida final del modelo: {final_loss:.4f}')

# Almacenar precisión
algorithm_8_loss = final_loss
```

```
Pérdida final del modelo: 3.6814
```

Fuente: Elaboración propia

En la figura Nº 115, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC, AUC, especificidad y recall, en donde se visualiza el código y el rendimiento de cada métrica. En donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, multi_class=”ovr” sirvió para las 3 clases, ya que el problema se descompone en múltiples subproblemas de clasificación binaria.

- **Redes Neuronales**

Figura Nº 116: Hiperparámetros - RN

```
param_grid = {
    'batch_size': [32, 64],
    'epochs': [10, 20],
    'dropout_rate': [0.2, 0.5],
    'learning_rate': [0.001, 0.01],
}

# Entrenar el modelo
model = MyKerasClassifier(build_fn=create_model, epochs=10, batch_size=32, verbose=0)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=3)
grid_result = grid.fit(X_train, y_train_encoded)

# Obtener las predicciones y probabilidades
y_pred = grid_result.predict(X_test)
y_prob = grid_result.predict_proba(X_test)
```

Fuente: Elaboración propia

Parámetros a explorar

Se definieron los parámetros batch_size, epochs, dropout_rate y learning_rate. Se procedió a realizar la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 117: Evaluación de las métricas - RN

```
# Invertir la codificación de las etiquetas predichas
y_pred_labels = label_encoder.inverse_transform(tf.argmax(y_pred_probs, axis=1).numpy())

# Calcular y mostrar las métricas
accuracy = accuracy_score(y_test, y_pred_labels)
precision = precision_score(y_test, y_pred_labels, average='macro')
recall = recall_score(y_test, y_pred_labels, average='macro')
f1 = f1_score(y_test, y_pred_labels, average='macro')
mcc = matthews_corrcoef(y_test, y_pred_labels)
auc = roc_auc_score(pd.get_dummies(y_test), y_pred_probs, multi_class='ovr')

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'MCC: {mcc}')
print(f'AUC: {auc}')

# Calcular la pérdida (log loss)
cross = log_loss(y_test_encoded, y_pred_probs)
print("Entropía Cruzada Categórica:", cross)
```

Fuente: Elaboración propia

En la figura Nº 116, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC, AUC, recall y la entropía cruzada (fórmula para multiclase de la pérdida), en donde se visualiza el código y el rendimiento de cada métrica.

Figura Nº 117: Evaluación de las métricas - RN

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_7_spe = specificity
```

```
Specificity for class 0: 0.98
Specificity for class 1: 0.86
Specificity for class 2: 0.96
Average Specificity: 0.93
```

```
# Pérdida final del modelo
final_loss = history.history['loss'][-1]
print(f'Pérdida final del modelo: {final_loss:.4f}')

# Almacenar precisión
algorithm_7_loss = final_loss
```

```
Pérdida final del modelo: 3.4090
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la función de la pérdida, se imprimió el valor de la pérdida final que se tuvo en el modelo de redes neuronales, ya que durante el entrenamiento se visualiza el valor de perdida, empieza con una mayor perdida y termina con un valor de perdida bajo. El modelo de redes neuronales presenta cambios cuando se vuelve entrenar.

- **Regresión Logística**

Figura Nº 118: Hiperparámetros - RL

```
# Definir hiperparámetros a optimizar
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# Crear un modelo de regresión logística
model = LogisticRegression()

# Configurar GridSearchCV para buscar los mejores hiperparámetros
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
# Obtener el mejor modelo después de la búsqueda de hiperparámetros
best_model = grid_search.best_estimator_
```

```
# Mostrar los mejores hiperparámetros encontrados
print("Mejores hiperparámetros:", grid_search.best_params_)
```

```
Mejores hiperparámetros: {'C': 10}
```

Fuente: Elaboración propia

Parámetros a explorar

Se definió la variable parámetros, para guardar los parámetros. Se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 119: Evaluación de las métricas - RL

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score

# Calcular y mostrar la precisión
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo: {accuracy}')

# Almacenar precisión
algorithm_9_accuracy = accuracy
```

```
Precisión del modelo: 0.8548387096774194
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred, average='macro')
print("Precisión del modelo:", precision)

# Almacenar precisión
algorithm_9_precision = precision
```

```
Precisión del modelo: 0.7983193277310924
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred, average='macro')
print("Precisión del modelo:", f1)

# Almacenar precisión
algorithm_9_f1 = f1
```

```
Precisión del modelo: 0.8006349206349207
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, y_pred)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_9_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.7673757741443372
```

```
# Recall
recall = recall_score(y_test, y_pred, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_9_recall = recall
```

```
Recall: 0.8109549441786283
```

Fuente: Elaboración propia

En la figura Nº 118, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC y recall, en donde se visualiza el código y el rendimiento de cada métrica. También se almacenó cada métrica para poder realizar las gráficas y así poder comparar el rendimiento de los modelos.

Figura Nº 119: Evaluación de las métricas - RL

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_9_spe = specificity
```

```
Specificity for class 0: 0.88
Specificity for class 1: 0.93
Specificity for class 2: 1.00
Average Specificity: 0.94
```

```
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize

# Crear y entrenar el modelo de regresión logística
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred_proba = logistic_model.predict_proba(X_test)

# Binarizar las etiquetas del conjunto de prueba
y_true_binary = label_binarize(y_test, classes=logistic_model.classes_)

# Calcular el AUC usando la estrategia micro-average
auc_micro = roc_auc_score(y_true_binary, y_pred_proba, average='micro')

# Calcular el AUC usando la estrategia macro-average
auc_macro = roc_auc_score(y_true_binary, y_pred_proba, average='macro')

# Imprimir el AUC para micro-average y macro-average
print(f'AUC (Micro-average): {auc_micro:.2f}')
print(f'AUC (Macro-average): {auc_macro:.2f}')

# Almacenar precisión
algorithm_9_auc = auc_macro
```

```
AUC (Micro-average): 0.98
AUC (Macro-average): 0.97
```

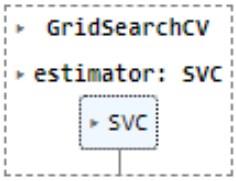
Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **Support Vector Machine**

Figura Nº 120: Hiperparámetros - SVM

```
# Configurar la búsqueda de cuadrícula con validación cruzada
grid_search = GridSearchCV(svm_model, param_grid, cv=5)

# Entrenar el modelo con la búsqueda de cuadrícula
grid_search.fit(X_train, y_train)
```




```
# Obtener los mejores hiperparámetros encontrados
best_params = grid_search.best_params_
print(f"Mejores hiperparámetros: {best_params}")
```

```
Mejores hiperparámetros: {'C': 1, 'kernel': 'linear'}
```

Fuente: Elaboración propia

Parámetros a explorar

Primeramente, se configuro la búsqueda de cuadrícula y se realizó la optimización de los hiperparámetros mediante GridSearchCV, se obtuvo los mejores hiperparámetros (c, kernel) para poder obtener el mejor modelo y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 121: Evaluación de las métricas - SVM

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Calcular y mostrar la precisión
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo: {accuracy}')

# Almacenar precisión
algorithm_10_accuracy = accuracy
```

```
Precisión del modelo: 0.8709677419354839
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_10_precision = precision
```

```
Precisión del modelo de stacking: 0.8148148148148149
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_10_f1 = f1
```

```
Precisión del modelo de stacking: 0.8148148148148149
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, y_pred)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_10_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.7891156462585034
```

```
# Recall
recall = recall_score(y_test, y_pred, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_10_recall = recall
```

```
Recall: 0.8148148148148149
```

Fuente: Elaboración propia

En la figura Nº 120, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC y recall, en donde se visualiza el código y el rendimiento de cada métrica. También se almacenó cada métrica para poder realizar las gráficas y así poder comparar el rendimiento de los modelos.

Figura Nº 121: Evaluación de las métricas - SVM

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_10_spe = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.91
Specificity for class 2: 1.00
Average Specificity: 0.94
```

```
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize

# Crear y entrenar el modelo SVM
svm_model = SVC(probability=True)
svm_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred_proba = svm_model.predict_proba(X_test)

# Binarizar las etiquetas del conjunto de prueba
y_true_binary = label_binarize(y_test, classes=svm_model.classes_)

# Calcular el AUC usando la estrategia micro-average
auc_micro = roc_auc_score(y_true_binary, y_pred_proba, average='micro')

# Calcular el AUC usando la estrategia macro-average
auc_macro = roc_auc_score(y_true_binary, y_pred_proba, average='macro')

# Imprimir el AUC para micro-average y macro-average
print(f'AUC (Micro-average): {auc_micro:.2f}')
print(f'AUC (Macro-average): {auc_macro:.2f}')

# Almacenar precisión
algorithm_10_auc = auc_macro
```

```
AUC (Micro-average): 0.96
AUC (Macro-average): 0.95
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **XGBoost**

Figura Nº 122: Hiperparámetros - XGB

```
# Definir el espacio de búsqueda de hiperparámetros
param_grid = {
    'eta': [0.1, 0.2, 0.3],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}
```

```
# Realizar la búsqueda en cuadrícula
grid_search = GridSearchCV(xgb_model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train_encoded)
```

```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

```
# Mostrar los mejores hiperparámetros
print("Mejores hiperparámetros:")
print(grid_search.best_params_)
```

```
Mejores hiperparámetros:
{'colsample_bytree': 1.0, 'eta': 0.2, 'max_depth': 4, 'subsample': 1.0}
```

Fuente: Elaboración propia

Parámetros a explorar

Primeramente, se estableció los hiperparámetros para la búsqueda de cuadrícula, se usaron los hiperparámetros eta, max_depth, subsample y colsample_bytree. Se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 123: Evaluación de las métricas - XGB

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Evaluar la precisión del modelo optimizado
accuracy = accuracy_score(y_test_encoded, y_pred_optimized)
print(f'Accuracy del modelo optimizado: {accuracy}')

# Almacenar precisión
algorithm_11_accuracy = accuracy
```

```
Accuracy del modelo optimizado: 0.8870967741935484
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test_encoded, y_pred_optimized, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_11_precision = precision
```

```
Precisión del modelo de stacking: 0.8417366946778712
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test_encoded, y_pred_optimized, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_11_f1 = f1
```

```
Precisión del modelo de stacking: 0.8458201058201058
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test_encoded, y_pred_optimized)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_11_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.81974484183498
```

```
# Recall
recall = recall_score(y_test_encoded, y_pred_optimized, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_11_recall = recall
```

```
Recall: 0.8588018341307815
```

Fuente: Elaboración propia

En la figura Nº 122, se visualiza la evaluación de las métricas de Acuraccy, precisión, f1-score, MCC y recall, en donde se visualiza el código y el rendimiento de cada métrica. También se almacenó cada métrica para poder realizar las gráficas y así poder comparar el rendimiento de los modelos.

Figura Nº 123: Evaluación de las métricas - XGB

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_11_spe = specificity
```

```
Specificity for class 0: 0.90
Specificity for class 1: 0.95
Specificity for class 2: 1.00
Average Specificity: 0.95
```

```
from sklearn.metrics import roc_auc_score

# Obtener las probabilidades de predicción para el conjunto de prueba
y_prob_optimized = grid_search.predict_proba(X_test)

# Calcular el AUC-ROC para un problema de clasificación multiclase
auc_roc = roc_auc_score(y_test_encoded, y_prob_optimized, multi_class='ovr', average='macro')

print(f'AUC-ROC del modelo optimizado: {auc_roc}')

# Almacenar precisión
algorithm_11_auc = auc_roc
```

```
AUC-ROC del modelo optimizado: 0.9594337790752219
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **AdaBoost**

Figura Nº 124: Hiperparámetros - ADA

```
# Definir la cuadrícula de hiperparámetros para buscar
param_grid = {
    'n_estimators': [50, 100, 150],
    'estimator__max_depth': [1, 2, 3],  # Usar 'estimator__' en lugar de 'base_classifier__'
    'learning_rate': [0.01, 0.1, 1.0]
}
```

```
# Realizar la búsqueda en cuadrícula
grid_search = GridSearchCV(adaboost_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
estimator: AdaBoostClassifier
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

```
# Obtener el mejor modelo y sus hiperparámetros
best_adaboost = grid_search.best_estimator_
best_params = grid_search.best_params_
```

```
# Imprimir los mejores hiperparámetros
print("Mejores hiperparámetros:", best_params)
```

```
Mejores hiperparámetros: {'estimator__max_depth': 3, 'learning_rate': 0.01, 'n_estimators': 100}
```

Fuente: Elaboración propia

Se definió los hiperparámetros n_estimators, estimator_max_depth y learning_rate. Luego se realizó la optimización de los hiperparámetros mediante GridSearchCV, y se obtuvo los mejores valores para poder obtener el mejor modelo con los mejores hiperparámetros que se hayan encontrado en la búsqueda y así para proceder a realizar las predicciones en el conjunto de prueba.

Evaluación de las métricas

Figura Nº 125: Evaluación de las métricas - ADA

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Calcular y mostrar la precisión
accuracy = accuracy_score(y_test, y_pred)
print(f'Acuraccy del modelo: {accuracy}')

# Almacenar precisión
algorithm_12_accuracy = accuracy
```

```
Acuraccy del modelo: 0.8548387096774194
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_12_precision = precision
```

```
Precisión del modelo de stacking: 0.7983193277310924
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_12_f1 = f1
```

```
Precisión del modelo de stacking: 0.8006349206349207
```

```
# Coeficiente de correlación de Matthews (MCC) para clasificación multiclase
mcc = matthews_corrcoef(y_test, y_pred)
print("Coeficiente de correlación de Matthews (MCC):", mcc)

# Almacenar precisión
algorithm_12_mcc = mcc
```

```
Coeficiente de correlación de Matthews (MCC): 0.7673757741443372
```

```
# Recall
recall = recall_score(y_test, y_pred, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_12_recall = recall
```

```
Recall: 0.8109549441786283
```

```
# Crear y entrenar el modelo AdaBoost con árbol de decisión como clasificador base
base_classifier = DecisionTreeClassifier(max_depth=1)
adaboost_model = AdaBoostClassifier(base_classifier, n_estimators=50, algorithm='SAMME.R')
adaboost_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred_proba = adaboost_model.predict_proba(X_test)

# Calcular el AUC utilizando la estrategia "macro-average"
y_true_binary = (y_test == 'Hernia')  # Elije una clase como referencia para la binarización
auc_macro = roc_auc_score(y_true_binary, y_pred_proba[:, 0], multi_class='ovr', average='macro')

# Imprimir el AUC para las tres clases combinadas
print(f'AUC (Macro-average): {auc_macro:.2f}')

# Almacenar precisión
algorithm_12_auc = auc_macro
```

```
AUC (Macro-average): 0.87
```

Fuente: Elaboración propia

Figura Nº 125: Evaluación de las métricas - ADA

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_12_spe = specificity
```

```
Specificity for class 0: 0.88
Specificity for class 1: 0.93
Specificity for class 2: 1.00
Average Specificity: 0.94
```

```
# Crear y entrenar el modelo AdaBoost con árbol de decisión como clasificador base
base_classifier = DecisionTreeClassifier(max_depth=1)
adaboost_model = AdaBoostClassifier(base_classifier, n_estimators=50, algorithm='SAMME.R')
adaboost_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred_proba = adaboost_model.predict_proba(X_test)

# Calcular el AUC para cada clase utilizando la estrategia "uno contra todos"
classes = adaboost_model.classes_
auc_per_class = {}

for i in range(len(classes)):
    y_true_binary = (y_test == classes[i])
    y_pred_proba_class = y_pred_proba[:, i]
    auc_per_class[classes[i]] = roc_auc_score(y_true_binary, y_pred_proba_class)

# Imprimir el AUC para cada clase
for class_label, auc_value in auc_per_class.items():
    print(f'AUC para {class_label}: {auc_value:.2f}')
```

```
AUC para Hernia: 0.87
AUC para Normal: 0.89
AUC para Spondylolisthesis: 1.00
```

Fuente: Elaboración propia

Se visualiza el resultado de cada métrica que fue utilizada, en donde se tomó en cuenta los valores de (y_test) y (y_pred) para la evaluación de cada métrica. En la métrica de especificidad se toman los valores que fueron hallados en la matriz de confusión y se imprime los valores de especificidad por clase y el promedio entre las clases. En la métrica de AUC, fue necesario binarizar las etiquetas del conjunto de prueba (y_test) y se halló los valores de AUC usando el promedio entre las clases (macro) y el (micro), pero solo se tomará en cuenta los valores de macro.

- **Stacking 1**

Figura Nº 126: Hiperparámetros – Stacking 1

```
[25]: # Particionando la data en 0.8 para entrenar y 0.2 para validar
      X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=3)

[26]: label_encoder = LabelEncoder()
      data["class"] = label_encoder.fit_transform(data["class"])

[27]: # Normalizar datos
      scaler = StandardScaler()
      X_train_scaled = scaler.fit_transform(X_train)
      X_test_scaled = scaler.transform(X_test)

[28]: # Definir modelos base
      models = [
          ('xgboost', XGBClassifier()),
          ('nb', GaussianNB()),
          ('svc', SVC(probability=True)),
          ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
      ]

[29]: # Definir meta-modelo
      meta_model = LogisticRegression()

[30]: # Crear modelo apilado
      stacked_model = StackingClassifier(estimators=models, final_estimator=meta_model)

[31]: # Entrenar el modelo apilado
      stacked_model.fit(X_train, y_train)

[31]: StackingClassifier
      xgboost: XGBClassifier
      nb: GaussianNB
      svc: SVC
      rf: RandomForestClassifier
      final_estimator: LogisticRegression

[32]: # Hacer predicciones
      y_pred_stacked = stacked_model.predict(X_test)
```

Fuente: Elaboración propia

Se visualiza que el primer modelo stacking está compuesto por los algoritmos XGBoost, Naive Bayes, Support Vector Machine y Random Forest. Se utilizo al algoritmo de Regresión Logística como meta-modelo. Se procedió a entrenar el modelo y así poder realizar la evaluación de las métricas.

Evaluación de las métricas

Figura Nº 127: Evaluación de las métricas – Stacking 1

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Evaluar el rendimiento
accuracy = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy}')

# Almacenar precisión
algorithm_01_accuracy = accuracy
```

```
Accuracy del modelo apilado: 0.8870967741935484
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_01_precision = precision
```

```
Precisión del modelo de stacking: 0.8418803418803419
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_01_f1 = f1
```

```
Precisión del modelo de stacking: 0.8483125983125984
```

```
import numpy as np
# Convertir las etiquetas a valores numéricos
label_encoder = LabelEncoder()
y_test_encoded = label_encoder.fit_transform(y_test)
y_pred_encoded = label_encoder.transform(y_pred_stacked)

# Calcular el MCC para cada clase
mcc_per_class = []

for class_label in np.unique(y_test_encoded):
    y_true_binary = (y_test_encoded == class_label).astype(int)
    y_pred_binary = (y_pred_encoded == class_label).astype(int)
    mcc_class = matthews_corrcoef(y_true_binary, y_pred_binary)
    mcc_per_class.append(mcc_class)

# Calcular el MCC total (promedio ponderado)
weights = np.bincount(y_test_encoded)
mcc_total = np.average(mcc_per_class, weights=weights)

print("MCC por clase:", mcc_per_class)
print("MCC total:", mcc_total)

# Almacenar precisión
algorithm_01_mcc = mcc_total
```

```
MCC por clase: [0.6942197261382851, 0.7309769912553625, 0.9682458365518543]
MCC total: 0.8469168805005219
```

Fuente: Elaboración propia

Se visualiza la evaluación y resultados de las métricas de Acuraccy, precisión, f1-score y MCC, en donde se obtuvo el resultado de cada clase, para así proceder a realizar el promedio de las 3 clases.

Figura Nº 127: Evaluación de las métricas – Stacking 1

```
# Recall
recall = recall_score(y_test, y_pred_stacked, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_01_recall = recall
```

```
Recall: 0.8588018341307815
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_01_specificity = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.93
Specificity for class 2: 1.00
Average Specificity: 0.95
```

```
import numpy as np
# Obtener clases únicas en y_test
unique_classes_test = np.unique(y_test)

# Hacer predicciones de probabilidades
y_prob_stacked = stacked_model.predict_proba(X_test_scaled)

# Calcular el AUC sin especificar labels
auc= roc_auc_score(y_test, y_prob_stacked, multi_class='ovr')
print(f'AUC del modelo apilado: {auc}')

# Almacenar precisión
algorithm_01_auc = auc
```

```
AUC del modelo apilado: 0.447967818530179
```

```
# guardar modelo
```

```
import pickle
filename = 'stack1_model.sav'
pickle.dump(stacked_model, open(filename, 'wb'))
```

Fuente: Elaboración propia

Se visualiza los resultados de las métricas recall, especificidad y AUC. También se observa el código que se usó para guardar los modelos del sistema inteligente.

- **Stacking 2**

Figura Nº 128: Hiperparámetros – Stacking 2

```python
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Cargar datos
data = pd.read_csv('column_3C.csv')

# Seleccionar características y etiquetas
X = data[['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis']].values
Y = data['class']

# Particionando la data en 0.8 para entrenar y 0.2 para validar
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=3)

# Normalizar datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Definir modelos base
models = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('knn', KNeighborsClassifier()),
    ('nb', GaussianNB()),
    ('svm', SVC(probability=True)),
]

# Definir meta-modelo
meta_model = LogisticRegression()

# Crear modelo apilado
stacked_model = StackingClassifier(estimators=models, final_estimator=meta_model, cv=5)

# Entrenar el modelo apilado
stacked_model.fit(X_train, y_train)

# Hacer predicciones
y_pred_stacked = stacked_model.predict(X_test)

# Evaluar el rendimiento
accuracy_stacked = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy_stacked}')
```

```
Accuracy del modelo apilado: 0.8709677419354839
```

Fuente: Elaboración propia

Se visualiza que el segundo modelo stacking está compuesto por los algoritmos Random Forest, KNN, Naive Bayes y Support Vector Machine. Se utilizo al algoritmo de Regresión Logística como meta-modelo. Se procedió a entrenar el modelo y así poder realizar la evaluación de las métricas.

Evaluación de las métricas

Figura Nº 129: Evaluación de las métricas – Stacking 2

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Evaluar el rendimiento
accuracy_stacked = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy_stacked}')

# Almacenar precisión
algorithm_02_accuracy = accuracy_stacked
```

```
Accuracy del modelo apilado: 0.8709677419354839
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_02_precision = precision
```

```
Precisión del modelo de stacking: 0.8272604588394062
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_02_f1 = f1
```

```
Precisión del modelo de stacking: 0.8357385398981325
```

```
import numpy as np
from sklearn.preprocessing import LabelEncoder

# Convertir las etiquetas a valores numéricos
label_encoder = LabelEncoder()
y_test_encoded = label_encoder.fit_transform(y_test)
y_pred_encoded = label_encoder.transform(y_pred_stacked)

# Calcular el MCC para cada clase
mcc_per_class = []

for class_label in np.unique(y_test_encoded):
    y_true_binary = (y_test_encoded == class_label).astype(int)
    y_pred_binary = (y_pred_encoded == class_label).astype(int)
    mcc_class = matthews_corrcoef(y_true_binary, y_pred_binary)
    mcc_per_class.append(mcc_class)

# Calcular el MCC total (promedio ponderado)
weights = np.bincount(y_test_encoded)
mcc_total = np.average(mcc_per_class, weights=weights)

print("MCC por clase:", mcc_per_class)
print("MCC total:", mcc_total)

# Almacenar precisión
algorithm_02_mcc = mcc_total
```

```
MCC por clase: [0.6942197261382851, 0.6964504283965728, 0.9375]
MCC total: 0.820467340758968
```

Fuente: Elaboración propia

Se visualiza la evaluación y resultados de las métricas de Acuraccy, precisión, f1-score y MCC, en donde se obtuvo el resultado de cada clase, para así proceder a realizar el promedio de las 3 clases.

Figura Nº 129: Evaluación de las métricas – Stacking 2

```
# Recall
recall = recall_score(y_test, y_pred_stacked, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_02_recall = recall
```

```
Recall: 0.8483851674641149
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_02_specificity = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.91
Specificity for class 2: 1.00
Average Specificity: 0.94
```

```
import numpy as np
# Obtener clases únicas en y_test
unique_classes_test = np.unique(y_test)

# Hacer predicciones de probabilidades
y_prob_stacked = stacked_model.predict_proba(X_test_scaled)

# Calcular el AUC sin especificar labels
auc_stacked = roc_auc_score(y_test, y_prob_stacked, multi_class='ovr')
print(f'AUC del modelo apilado: {auc_stacked}')


# Almacenar precisión
algorithm_02_auc = auc_stacked
```

```
AUC del modelo apilado: 0.47826228526535425
```

Fuente: Elaboración propia

Se visualiza los resultados de las métricas recall, especificidad y AUC. Donde se halló la especificidad por cada clase (Hernia, Normal y Espondilolistesis) y así obtener el resultado promedio de las 3 clases.

- **Stacking 3**

Figura Nº 130: Hiperparámetros – Stacking 3

```
from sklearn.ensemble import StackingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Cargar datos
data = pd.read_csv('column_3C.csv')

# Seleccionar caracteristicas y etiquetas
X = data[['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis']].values
Y = data['class']

# Particionando la data en 0.8 para entrenar y 0.2 para validar
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=3)

# Normalizar datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Definir modelos base
models = [
    ('xgb', XGBClassifier()),
    ('svm', SVC(probability=True)),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('knn', KNeighborsClassifier()),
]

# Definir meta-modelo
meta_model = LogisticRegression()

# Crear modelo apilado
stacked_model = StackingClassifier(estimators=models, final_estimator=meta_model, cv=5)

# Entrenar el modelo apilado
stacked_model.fit(X_train, y_train)

# Hacer predicciones
y_pred_stacked = stacked_model.predict(X_test)

# Evaluar el rendimiento
accuracy_stacked = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy_stacked}')
```

```
Accuracy del modelo apilado: 0.8870967741935484
```

Fuente: Elaboración propia

Se visualiza que el tercer modelo stacking está compuesto por los algoritmos XGBoost, Support Vector Machine, Random Forest y KNN. Se utilizo al algoritmo de Regresión Logística como meta-modelo. Se procedió a entrenar el modelo y así poder realizar la evaluación de las métricas.

Evaluación de las métricas

Figura Nº 131: Evaluación de las métricas – Stacking 3

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Evaluar el rendimiento
accuracy_stacked = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy_stacked}')

# Almacenar precisión
algorithm_03_accuracy = accuracy_stacked
```

```
Accuracy del modelo apilado: 0.8870967741935484
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_03_precision = precision
```

```
Precisión del modelo de stacking: 0.8418803418803419
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_03_f1 = f1
```

```
Precisión del modelo de stacking: 0.8483125983125984
```

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
# Convertir las etiquetas a valores numéricos
label_encoder = LabelEncoder()
y_test_encoded = label_encoder.fit_transform(y_test)
y_pred_encoded = label_encoder.transform(y_pred_stacked)

# Calcular el MCC para cada clase
mcc_per_class = []

for class_label in np.unique(y_test_encoded):
    y_true_binary = (y_test_encoded == class_label).astype(int)
    y_pred_binary = (y_pred_encoded == class_label).astype(int)
    mcc_class = matthews_corrcoef(y_true_binary, y_pred_binary)
    mcc_per_class.append(mcc_class)

# Calcular el MCC total (promedio ponderado)
weights = np.bincount(y_test_encoded)
mcc_total = np.average(mcc_per_class, weights=weights)

print("MCC por clase:", mcc_per_class)
print("MCC total:", mcc_total)

# Almacenar precisión
algorithm_03_mcc = mcc_total
```

```
MCC por clase: [0.6942197261382851, 0.7309769912553625, 0.9682458365518543]
MCC total: 0.8469168805005219
```

Fuente: Elaboración propia

Se visualiza la evaluación y resultados de las métricas de Acuraccy, precisión, f1-score y MCC, en donde se obtuvo el resultado de cada clase, para así proceder a realizar el promedio de las 3 clases.

Figura Nº 131: Evaluación de las métricas – Stacking 3

```
# Recall
recall = recall_score(y_test, y_pred_stacked, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_03_recall = recall
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_03_specificity = specificity
```

```
import numpy as np
# Obtener clases únicas en y_test
unique_classes_test = np.unique(y_test)

# Hacer predicciones de probabilidades
y_prob_stacked = stacked_model.predict_proba(X_test_scaled)

# Calcular el AUC sin especificar labels
auc_stacked = roc_auc_score(y_test, y_prob_stacked, multi_class='ovr')
print(f'AUC del modelo apilado: {auc_stacked}')

# Almacenar precisión
algorithm_03_auc = auc_stacked
```

```
AUC del modelo apilado: 0.43633309115345265
```

Fuente: Elaboración propia

Se visualiza el código que fue usado para la obtención de los resultados de las métricas de recall, especificidad y AUC. Donde se halló la especificidad por cada clase (Hernia, Normal y Espondilolistesis) y así obtener el resultado promedio de las 3 clases.

- **Stacking 4**

Figura Nº 132: Evaluación de las métricas – Stacking 4

```
from sklearn.ensemble import StackingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Cargar datos
data = pd.read_csv('column_3C.csv')

# Seleccionar caracteristicas y etiquetas
X = data[['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis']].values
Y = data['class']

# Particionando la data en 0.8 para entrenar y 0.2 para validar
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=3)

label_encoder = LabelEncoder()
data["class"] = label_encoder.fit_transform(data["class"])

# Normalizar datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Definir modelos base
models = [
    ('xgb', XGBClassifier()),
    ('svm', SVC(probability=True)),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('adaboost', AdaBoostClassifier(n_estimators=50, random_state=42)),
]

# Definir meta-modelo
meta_model = LogisticRegression()

# Crear modelo apilado
stacked_model = StackingClassifier(estimators=models, final_estimator=meta_model, cv=5)

# Entrenar el modelo apilado
stacked_model.fit(X_train, y_train)

# Hacer predicciones
y_pred_stacked = stacked_model.predict(X_test)

# Evaluar el rendimiento
accuracy_stacked = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy_stacked}')
```

```
Accuracy del modelo apilado: 0.8870967741935484
```

Fuente: Elaboración propia

Se visualiza que el cuarto modelo stacking está compuesto por los algoritmos XGBoost, Support Vector Machine, Random Forest y AdaBoost. Se utilizo al algoritmo de Regresión Logística como meta-modelo. Se procedió a entrenar el modelo y así poder realizar la evaluación de las métricas.

Evaluación de las métricas

Figura Nº 133: Evaluación de las métricas – Stacking 4

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Evaluar el rendimiento
accuracy_stacked = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy_stacked}')

# Almacenar precisión
algorithm_04_accuracy = accuracy_stacked
```

```
Accuracy del modelo apilado: 0.8870967741935484
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_04_precision = precision
```

```
Precisión del modelo de stacking: 0.8418803418803419
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_04_f1 = f1
```

```
Precisión del modelo de stacking: 0.8483125983125984
```

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
# Convertir las etiquetas a valores numéricos
label_encoder = LabelEncoder()
y_test_encoded = label_encoder.fit_transform(y_test)
y_pred_encoded = label_encoder.transform(y_pred_stacked)

# Calcular el MCC para cada clase
mcc_per_class = []

for class_label in np.unique(y_test_encoded):
    y_true_binary = (y_test_encoded == class_label).astype(int)
    y_pred_binary = (y_pred_encoded == class_label).astype(int)
    mcc_class = matthews_corrcoef(y_true_binary, y_pred_binary)
    mcc_per_class.append(mcc_class)

# Calcular el MCC total (promedio ponderado)
weights = np.bincount(y_test_encoded)
mcc_total = np.average(mcc_per_class, weights=weights)

print("MCC por clase:", mcc_per_class)
print("MCC total:", mcc_total)

# Almacenar precisión
algorithm_04_mcc = mcc_total
```

```
MCC por clase: [0.6942197261382851, 0.7309769912553625, 0.9682458365518543]
MCC total: 0.8469168805005219
```

Fuente: Elaboración propia

Se visualiza la evaluación y resultados de las métricas de Acuraccy, precisión, f1-score y MCC, en donde se obtuvo el resultado de cada clase, para así proceder a realizar el promedio de las 3 clases.

Figura Nº 133: Evaluación de las métricas – Stacking 4

```
# Recall
recall = recall_score(y_test, y_pred_stacked, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_04_recall = recall
```

```
Recall: 0.8588018341307815
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_04_specificity = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.93
Specificity for class 2: 1.00
Average Specificity: 0.95
```

```
import numpy as np
# Obtener clases únicas en y_test
unique_classes_test = np.unique(y_test)

# Hacer predicciones de probabilidades
y_prob_stacked = stacked_model.predict_proba(X_test_scaled)

# Calcular el AUC sin especificar labels
auc_stacked = roc_auc_score(y_test, y_prob_stacked, multi_class='ovr')
print(f'AUC del modelo apilado: {auc_stacked}')

# Almacenar precisión
algorithm_04_auc = auc_stacked
```

```
AUC del modelo apilado: 0.502650315161115
```

Fuente: Elaboración propia

Se visualiza el código que fue usado para la obtención de los resultados de las métricas de recall, especificidad y AUC. Donde se halló la especificidad por cada clase (Hernia, Normal y Espondilolistesis) y así obtener el resultado promedio de las 3 clases.

- **Stacking 5**

Figura Nº 134: Evaluación de las métricas – Stacking 5

```python
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Cargar datos
data = pd.read_csv('column_3C.csv')

# Seleccionar caracteristicas y etiquetas
X = data[['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis']].values
Y = data['class']

# Convertir etiquetas a números (si es necesario)
# Puedes usar LabelEncoder o directamente el parámetro "dtype" de train_test_split
# Aquí asumiré que ya tienes etiquetas numéricas (0, 1, 2) para las clases.

# Particionar la data en 0.8 para entrenar y 0.2 para validar
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=3)

# Normalizar datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Definir modelos base
models = [
    ('decision_tree', DecisionTreeClassifier()),
    ('xgb', XGBClassifier()),
    ('svm', SVC(probability=True)),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
]

# Definir meta-modelo
meta_model = LogisticRegression(multi_class='multinomial', solver='lbfgs')

# Crear modelo apilado
stacked_model = StackingClassifier(estimators=models, final_estimator=meta_model, cv=5)

# Entrenar el modelo apilado
stacked_model.fit(X_train_scaled, y_train)

# Hacer predicciones
y_pred_stacked = stacked_model.predict(X_test_scaled)
```

Fuente: Elaboración propia

Se visualiza que el quinto modelo stacking está compuesto por los algoritmos Decision Tree, XGBoost, Support Vector Machine y Random Forest. Se utilizo al algoritmo de Regresión Logística como meta-modelo. Se procedió a entrenar el modelo y así poder realizar la evaluación de las métricas.

Evaluación de las métricas

Figura Nº 135: Evaluación de las métricas – Stacking 5

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score, f1_score, matthews_corrcoef, recall_score
```

```
# Evaluar el rendimiento
accuracy_stacked = accuracy_score(y_test, y_pred_stacked)
print(f'Accuracy del modelo apilado: {accuracy_stacked}')

# Almacenar precisión
algorithm_05_accuracy = accuracy_stacked
```

```
Accuracy del modelo apilado: 0.8870967741935484
```

```
# Precisión (Precision) para clasificación multiclase
precision = precision_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", precision)

# Almacenar precisión
algorithm_05_precision = precision
```

```
Precisión del modelo de stacking: 0.8943341604631927
```

```
# Puntuación F1 (F1-score) para clasificación multiclase
f1 = f1_score(y_test, y_pred_stacked, average='macro')
print("Precisión del modelo de stacking:", f1)

# Almacenar precisión
algorithm_05_f1 = f1
```

```
Precisión del modelo de stacking: 0.8894753047978854
```

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
# Convertir las etiquetas a valores numéricos
label_encoder = LabelEncoder()
y_test_encoded = label_encoder.fit_transform(y_test)
y_pred_encoded = label_encoder.transform(y_pred_stacked)

# Calcular el MCC para cada clase
mcc_per_class = []

for class_label in np.unique(y_test_encoded):
    y_true_binary = (y_test_encoded == class_label).astype(int)
    y_pred_binary = (y_pred_encoded == class_label).astype(int)
    mcc_class = matthews_corrcoef(y_true_binary, y_pred_binary)
    mcc_per_class.append(mcc_class)

# Calcular el MCC total (promedio ponderado)
weights = np.bincount(y_test_encoded)
mcc_total = np.average(mcc_per_class, weights=weights)

print("MCC por clase:", mcc_per_class)
print("MCC total:", mcc_total)

# Almacenar precisión
algorithm_05_mcc = mcc_total
```

```
Coeficiente de correlación de Matthews (MCC): 0.8177313541168356
```

Fuente: Elaboración propia

Se visualiza la evaluación y resultados de las métricas de Acuraccy, precisión, f1-score y MCC, en donde se obtuvo el resultado de cada clase, para así proceder a realizar el promedio de las 3 clases.

Figura Nº 135: Evaluación de las métricas – Stacking 5

```
# Recall
recall = recall_score(y_test, y_pred_stacked, average='macro')
print("Recall:", recall)

# Almacenar precisión
algorithm_05_recall = recall
```

```
Recall: 0.8870967741935484
```

```
# Calcular la especificidad para cada clase
num_classes = len(conf_matrix)
specificities = []

for i in range(num_classes):
    tn = sum(conf_matrix[j, j] for j in range(num_classes) if j != i)
    fp = sum(conf_matrix[j, i] for j in range(num_classes) if j != i)
    specificity = tn / (tn + fp)
    specificities.append(specificity)
    print(f'Specificity for class {i}: {specificity:.2f}')

# Calcular la especificidad promedio
specificity = sum(specificities) / num_classes
print(f'Average Specificity: {specificity:.2f}')

# Almacenar precisión
algorithm_05_specificity = specificity
```

```
Specificity for class 0: 0.92
Specificity for class 1: 0.93
Specificity for class 2: 1.00
Average Specificity: 0.95
```

```
import numpy as np
# Obtener clases únicas en y_test
unique_classes_test = np.unique(y_test)

# Hacer predicciones de probabilidades
y_prob_stacked = stacked_model.predict_proba(X_test_scaled)

# Calcular el AUC sin especificar labels
auc_stacked = roc_auc_score(y_test, y_prob_stacked, multi_class='ovr')
print(f'AUC del modelo apilado: {auc_stacked}')

# Almacenar precisión
algorithm_05_auc = auc_stacked
```

```
AUC del modelo apilado: 0.9693908857820047
```

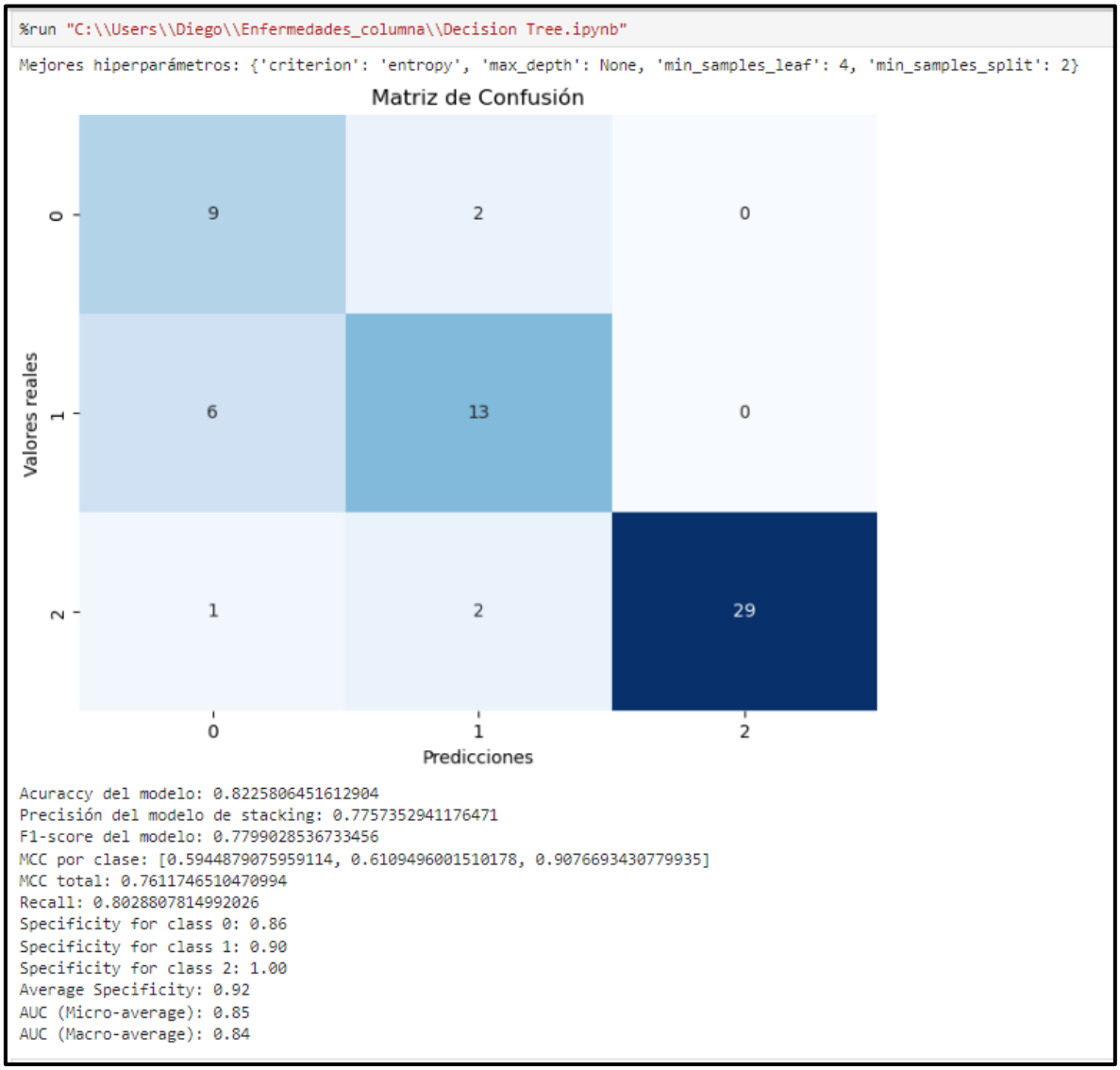
Fuente: Elaboración propia

Se visualiza el código que fue usado para la obtención de los resultados de las métricas de recall, especificidad y AUC. Donde se halló la especificidad por cada clase (Hernia, Normal y Espondilolistesis) y así obtener el resultado promedio de las 3 clases.

Quinta etapa: Interpretación

- **Decision Tree**

Figura Nº 136: Reporte de resultados del algoritmo – DT



Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Decision Tree, también se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases. Por último, se visualiza los valores de AUC macro y micro.

- **Gradient Boosting**

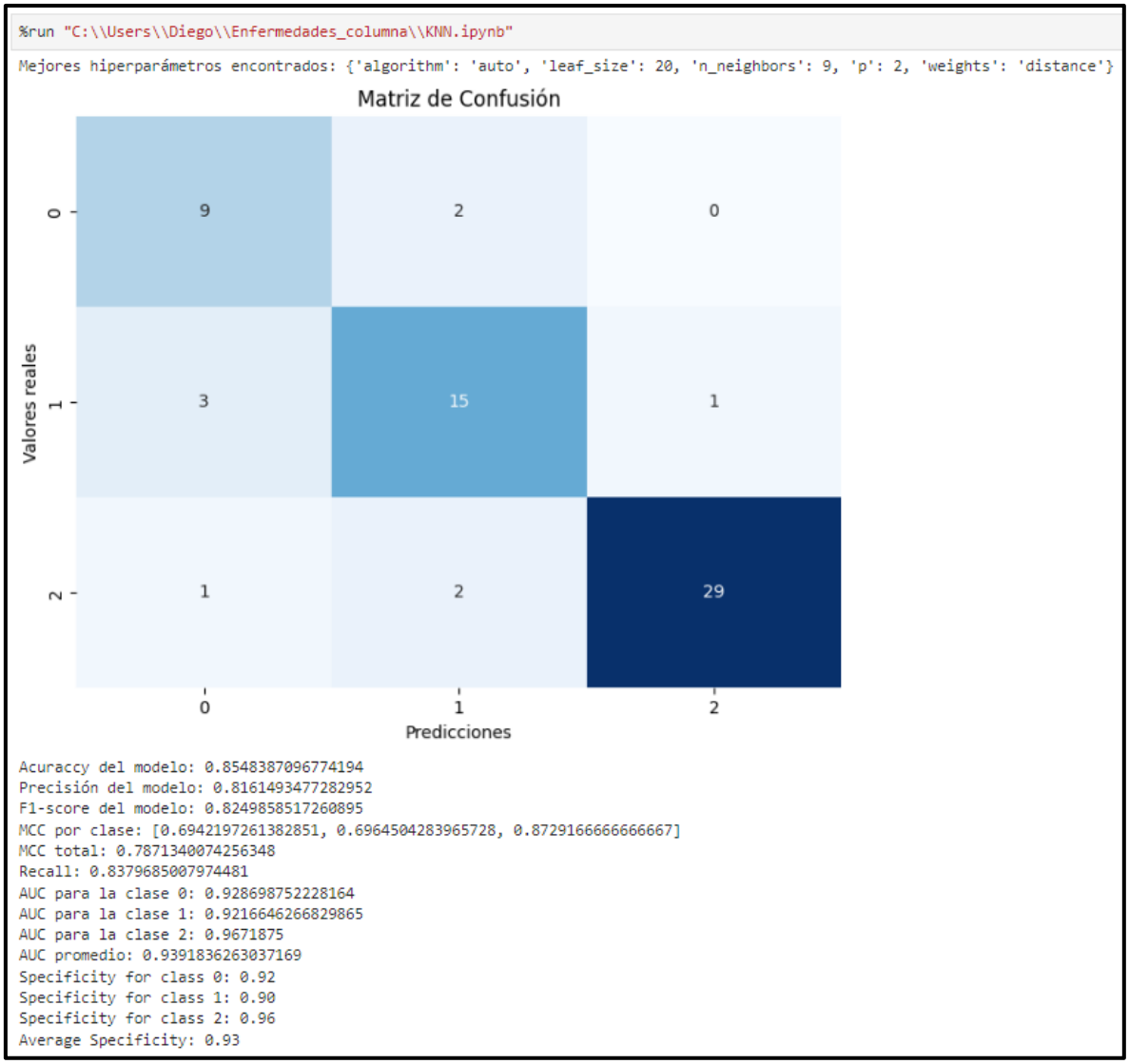
Figura Nº 137: Reporte de resultados del algoritmo – GB

Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Gradient Boosting, también se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases. Por último, se visualiza el resultado de AUC.

- **K-Nearest Neighbors**

Figura Nº 138: Reporte de resultados del algoritmo – KNN

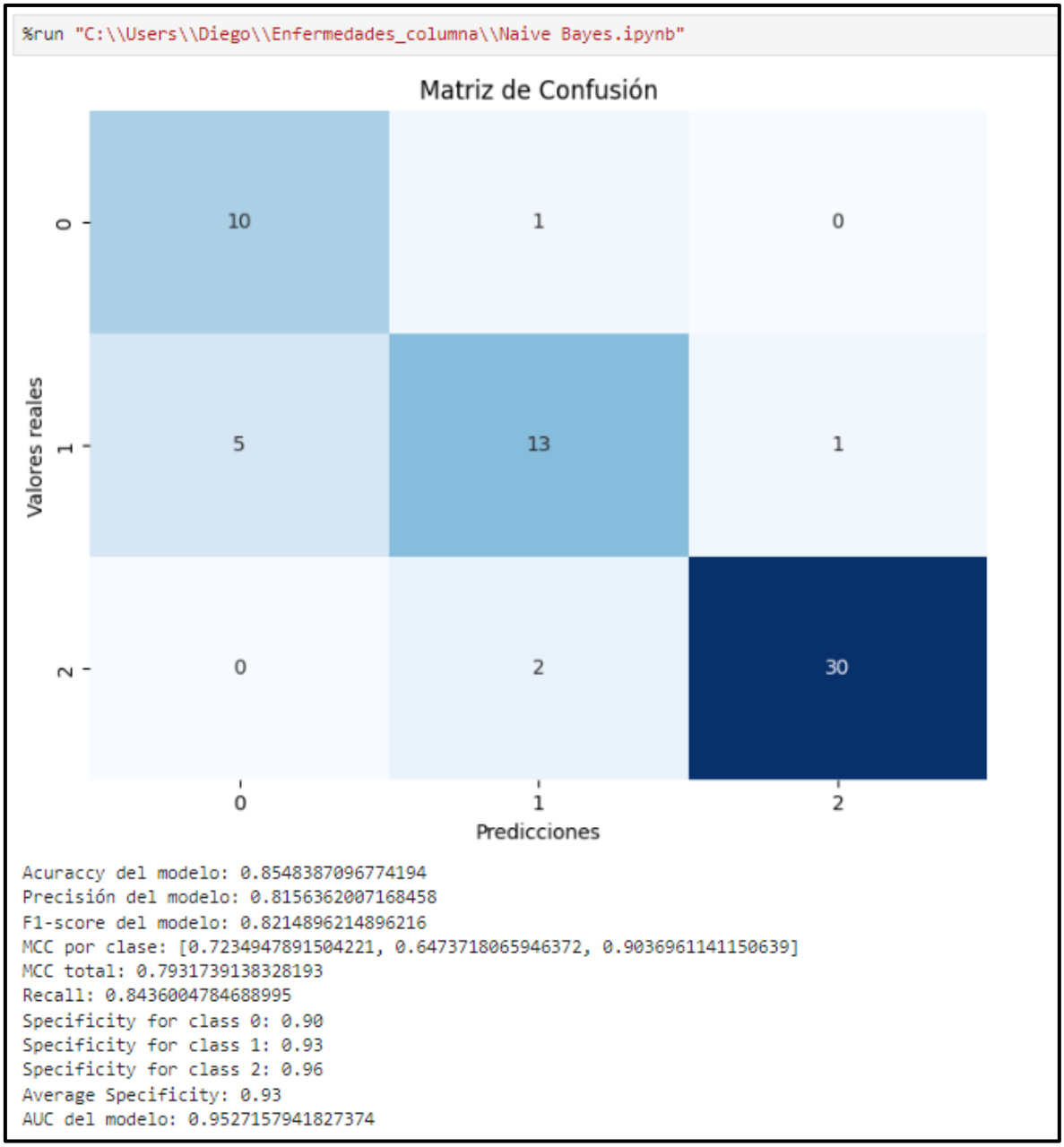


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo k-Nearest Neighbors también se observa los resultados de MCC, AUC y la especificidad de cada clase, para así obtener el resultado promedio de MCC, AUC y especificidad en función de las 3 clases.

- **Naive Bayes**

Figura Nº 139: Reporte de resultados del algoritmo – NB

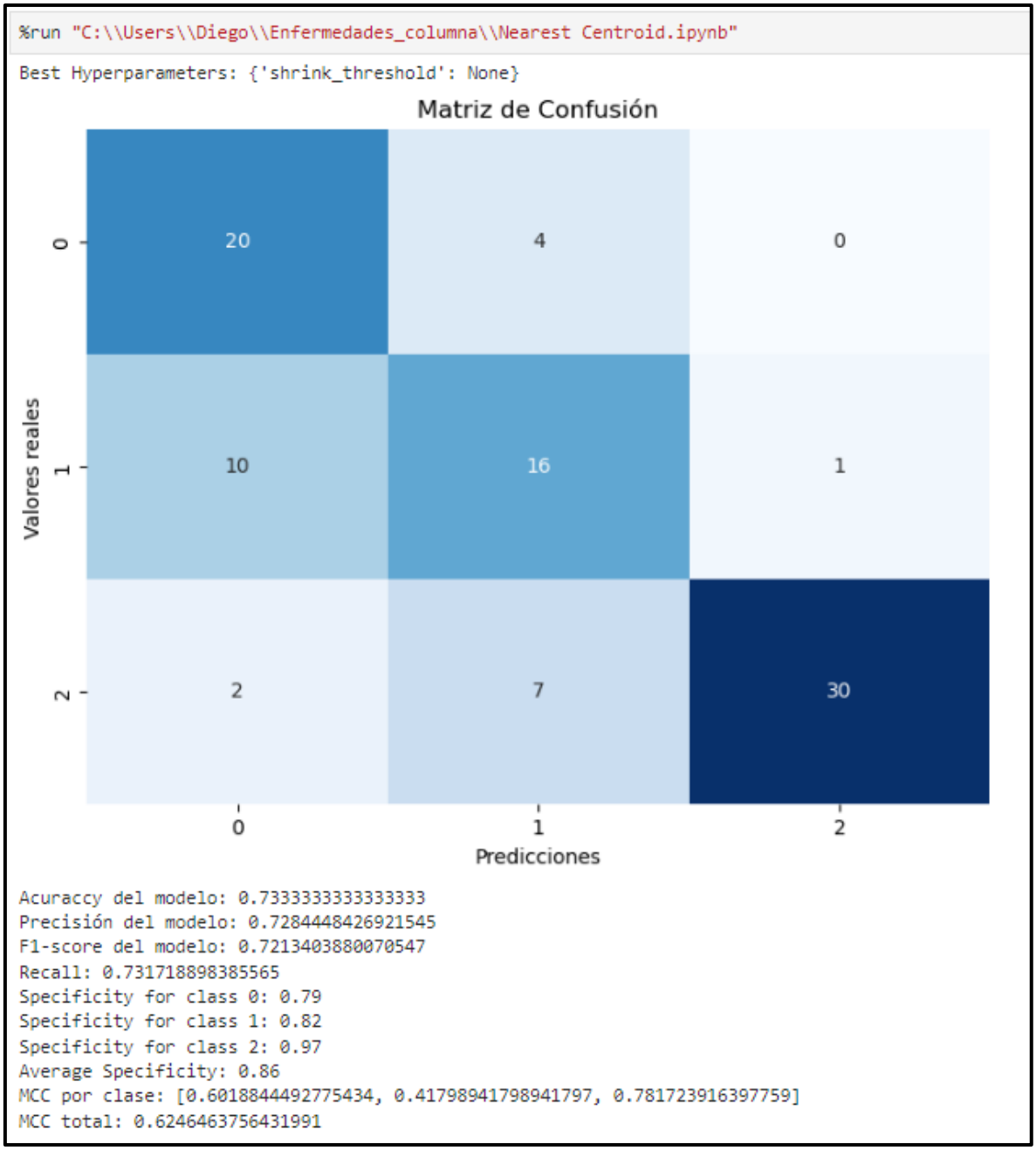


Fuente: Elaboración propia

Se visualiza el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Naive Bayes, también se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases. Por último, se visualiza el resultado de AUC del modelo Naive Bayes.

- **Nearest Centroid**

Figura Nº 140: Reporte de resultados del algoritmo – NC

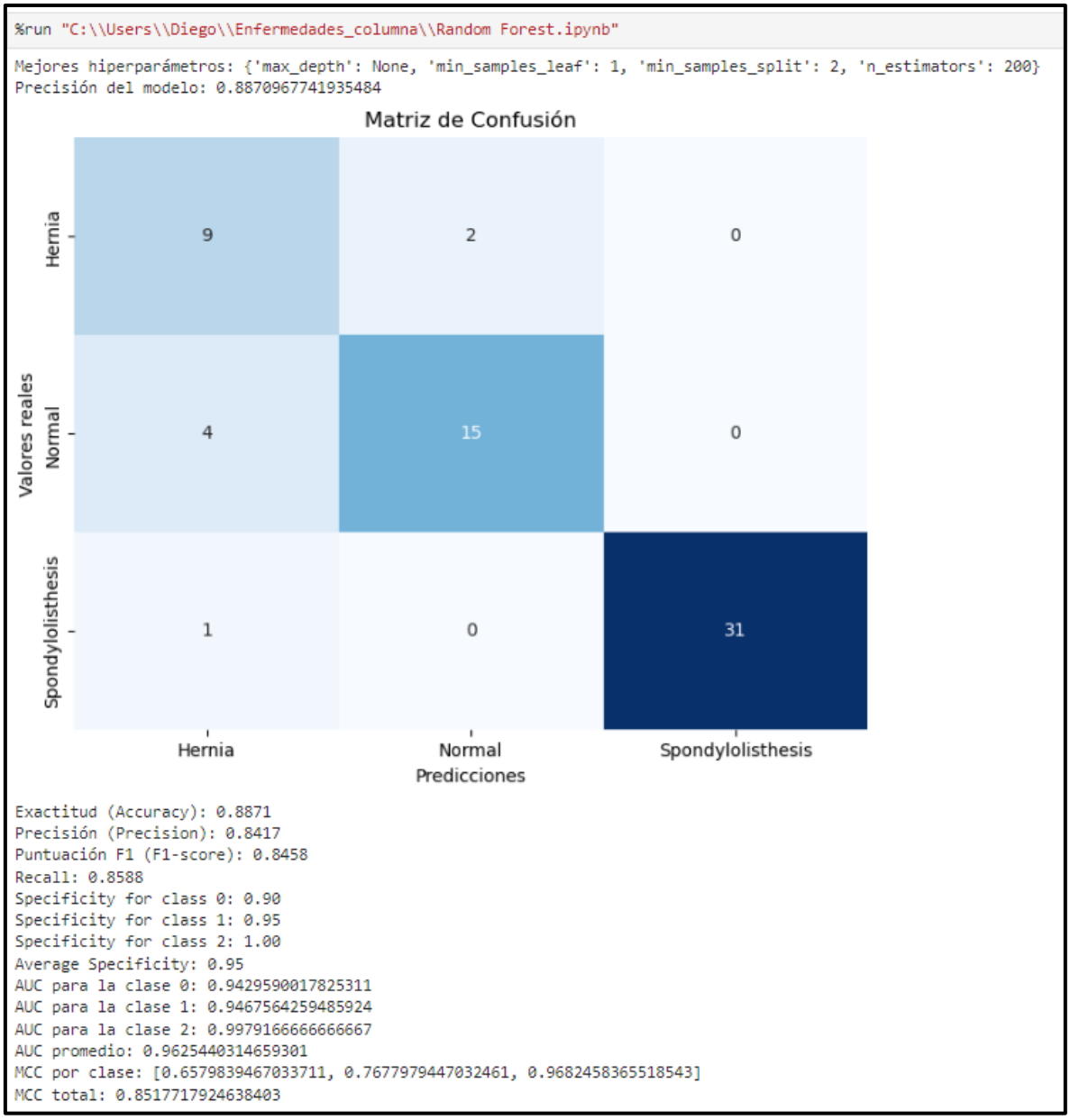


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Nearest Centroid, también se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Random Forest**

Figura Nº 141: Reporte de resultados del algoritmo – RF



Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Random Forest, también se observa los resultados de MCC, AUC y la especificidad de cada clase, para así obtener el resultado promedio de MCC, AUC y especificidad en función de las 3 clases.

- **Redes Neuronales**

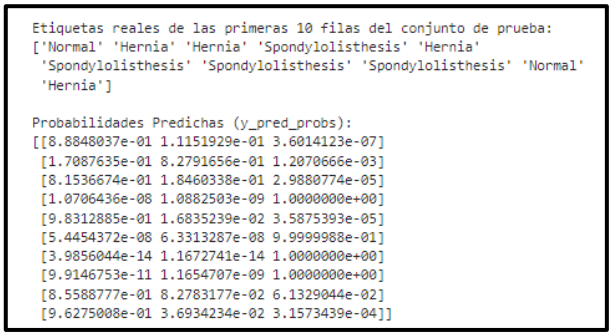
Figura Nº 142: Reporte de entrenamiento del algoritmo – RN

```
Mejores hiperparámetros: <keras_tuner.src.engine.hyperparameters.hyperparameters.HyperParameters object at 0x000001C8C8293410>
Epoch 1/30
5/5 [==============================] - 1s 53ms/step - loss: 9.6469 - accuracy: 0.3852 - val_loss: 7.9307 - val_accuracy: 0.0000e+0
Epoch 2/30
5/5 [==============================] - 0s 8ms/step - loss: 6.3854 - accuracy: 0.4346 - val_loss: 2.3502 - val_accuracy: 0.0704
Epoch 3/30
5/5 [==============================] - 0s 8ms/step - loss: 6.1219 - accuracy: 0.4064 - val_loss: 0.9067 - val_accuracy: 0.6338
Epoch 4/30
5/5 [==============================] - 0s 8ms/step - loss: 6.3726 - accuracy: 0.4134 - val_loss: 0.9663 - val_accuracy: 0.4789
Epoch 5/30
5/5 [==============================] - 0s 9ms/step - loss: 5.0684 - accuracy: 0.4700 - val_loss: 1.0312 - val_accuracy: 0.4366
Epoch 6/30
5/5 [==============================] - 0s 8ms/step - loss: 4.8227 - accuracy: 0.4558 - val_loss: 1.4733 - val_accuracy: 0.3662
Epoch 7/30
5/5 [==============================] - 0s 7ms/step - loss: 4.6839 - accuracy: 0.5159 - val_loss: 2.3772 - val_accuracy: 0.3521
Epoch 8/30
5/5 [==============================] - 0s 7ms/step - loss: 4.2589 - accuracy: 0.5265 - val_loss: 2.6140 - val_accuracy: 0.4366
Epoch 9/30
5/5 [==============================] - 0s 8ms/step - loss: 4.3006 - accuracy: 0.5406 - val_loss: 2.1437 - val_accuracy: 0.4648
Epoch 10/30
5/5 [==============================] - 0s 8ms/step - loss: 3.6650 - accuracy: 0.5548 - val_loss: 1.2901 - val_accuracy: 0.4789
Epoch 11/30
5/5 [==============================] - 0s 8ms/step - loss: 3.3388 - accuracy: 0.5512 - val_loss: 0.8289 - val_accuracy: 0.6197
Epoch 12/30
5/5 [==============================] - 0s 8ms/step - loss: 3.5701 - accuracy: 0.5689 - val_loss: 0.7098 - val_accuracy: 0.6620
Epoch 13/30
5/5 [==============================] - 0s 9ms/step - loss: 3.3225 - accuracy: 0.5371 - val_loss: 0.6855 - val_accuracy: 0.6620
Epoch 14/30
5/5 [==============================] - 0s 9ms/step - loss: 3.0462 - accuracy: 0.5548 - val_loss: 0.6951 - val_accuracy: 0.6761
Epoch 15/30
5/5 [==============================] - 0s 8ms/step - loss: 2.5486 - accuracy: 0.6113 - val_loss: 0.8381 - val_accuracy: 0.5634
Epoch 16/30
5/5 [==============================] - 0s 8ms/step - loss: 3.0096 - accuracy: 0.5724 - val_loss: 0.9645 - val_accuracy: 0.5211
Epoch 17/30
5/5 [==============================] - 0s 7ms/step - loss: 3.2125 - accuracy: 0.5618 - val_loss: 1.1456 - val_accuracy: 0.4789
Epoch 18/30
5/5 [==============================] - 0s 8ms/step - loss: 2.5741 - accuracy: 0.6078 - val_loss: 1.2381 - val_accuracy: 0.4789
Epoch 19/30
5/5 [==============================] - 0s 8ms/step - loss: 2.5376 - accuracy: 0.6290 - val_loss: 1.5099 - val_accuracy: 0.4789
Epoch 20/30
5/5 [==============================] - 0s 8ms/step - loss: 2.7861 - accuracy: 0.5760 - val_loss: 1.4733 - val_accuracy: 0.4930
Epoch 21/30
5/5 [==============================] - 0s 9ms/step - loss: 2.4854 - accuracy: 0.5972 - val_loss: 1.0744 - val_accuracy: 0.5070
Epoch 22/30
5/5 [==============================] - 0s 8ms/step - loss: 2.1614 - accuracy: 0.6219 - val_loss: 0.8344 - val_accuracy: 0.5634
Epoch 23/30
5/5 [==============================] - 0s 8ms/step - loss: 2.6847 - accuracy: 0.5901 - val_loss: 0.6932 - val_accuracy: 0.6338
Epoch 24/30
5/5 [==============================] - 0s 8ms/step - loss: 2.2824 - accuracy: 0.6078 - val_loss: 0.6140 - val_accuracy: 0.7324
Epoch 25/30
5/5 [==============================] - 0s 8ms/step - loss: 2.0923 - accuracy: 0.6148 - val_loss: 0.7674 - val_accuracy: 0.6056
Epoch 26/30
5/5 [==============================] - 0s 9ms/step - loss: 2.3877 - accuracy: 0.6325 - val_loss: 0.8793 - val_accuracy: 0.5634
Epoch 27/30
5/5 [==============================] - 0s 9ms/step - loss: 1.9861 - accuracy: 0.6643 - val_loss: 0.9811 - val_accuracy: 0.5352
Epoch 28/30
5/5 [==============================] - 0s 10ms/step - loss: 2.4494 - accuracy: 0.6466 - val_loss: 1.0194 - val_accuracy: 0.5352
Epoch 29/30
5/5 [==============================] - 0s 8ms/step - loss: 2.1228 - accuracy: 0.6572 - val_loss: 0.9388 - val_accuracy: 0.5493
Epoch 30/30
5/5 [==============================] - 0s 9ms/step - loss: 2.3050 - accuracy: 0.5936 - val_loss: 0.8035 - val_accuracy: 0.5775
2/2 [==============================] - 0s 3ms/step
```

Fuente: Elaboración propia

Se visualiza los mejores hiperparámetros y el entrenamiento del modelo de redes neuronales, aunque cuando el modelo es entrenado nuevamente, los resultados cambian. Por lo tanto, la figura Nº 141 representa a uno de los entrenamientos realizado en el modelo de redes neuronales.

Figura Nº 143: Probabilidades y etiquetas reales del algoritmo - RN

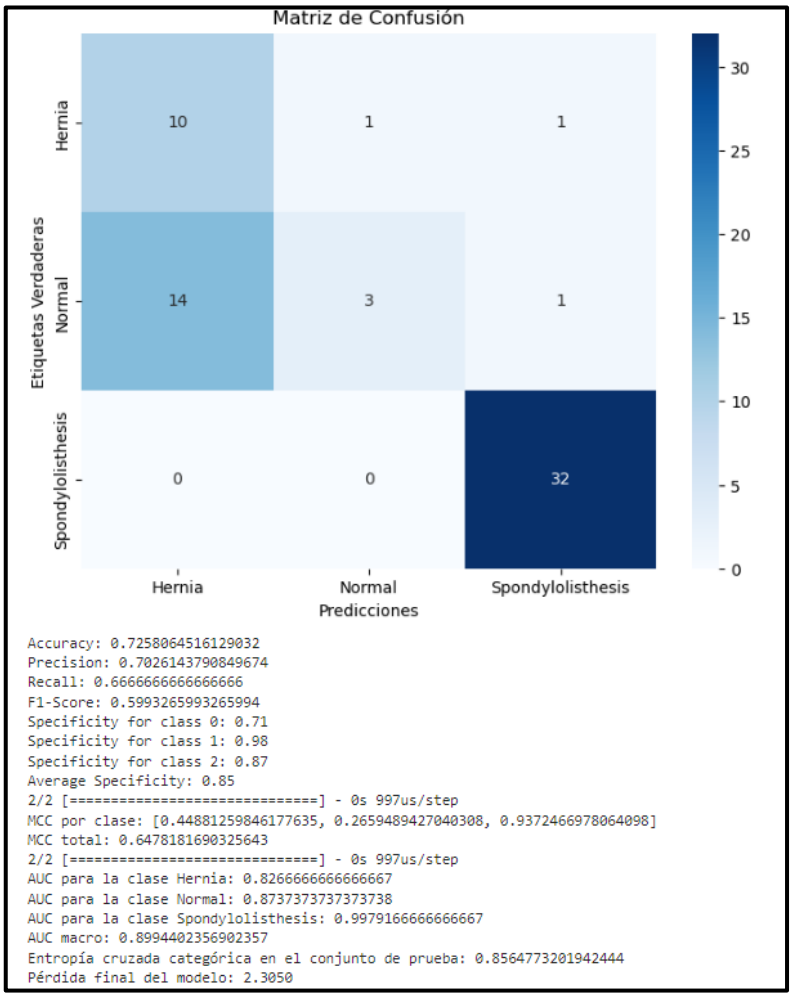


```
Etiquetas reales de las primeras 10 filas del conjunto de prueba:
['Normal' 'Hernia' 'Hernia' 'Spondylolisthesis' 'Hernia'
 'Spondylolisthesis' 'Spondylolisthesis' 'Spondylolisthesis' 'Normal'
 'Hernia']

Probabilidades Predichas (y_pred_probs):
[[8.8848037e-01 1.1151929e-01 3.6014123e-07]
 [1.7087635e-01 8.2791656e-01 1.2070666e-03]
 [8.1536674e-01 1.8460338e-01 2.9880774e-05]
 [1.0706436e-08 1.0882503e-09 1.0000000e+00]
 [9.8312885e-01 1.6835239e-02 3.5875393e-05]
 [5.4454372e-08 6.3313287e-08 9.9999988e-01]
 [3.9856044e-14 1.1672741e-14 1.0000000e+00]
 [9.9146753e-11 1.1654707e-09 1.0000000e+00]
 [8.5588777e-01 8.2783177e-02 6.1329044e-02]
 [9.6275008e-01 3.6934234e-02 3.1573439e-04]]
```

Fuente: Elaboración propia

Se visualiza los resultados de las etiquetas reales y las probabilidades predichas que se usaran para la aplicación de la fórmula de entropía categórica cruzada.

Figura Nº 144: Reporte de resultados del algoritmo – RN




```
Accuracy: 0.7258064516129032
Precision: 0.7026143790849674
Recall: 0.6666666666666666
F1-Score: 0.5993265993265994
Specificity for class 0: 0.71
Specificity for class 1: 0.98
Specificity for class 2: 0.87
Average Specificity: 0.85
2/2 [==============================] - 0s 997us/step
MCC por clase: [0.44881259846177635, 0.2659489427040308, 0.9372466978064098]
MCC total: 0.6478181690325643
2/2 [==============================] - 0s 997us/step
AUC para la clase Hernia: 0.8266666666666667
AUC para la clase Normal: 0.8737373737373738
AUC para la clase Spondylolisthesis: 0.9979166666666667
AUC macro: 0.8994402356902357
Entropía cruzada categórica en el conjunto de prueba: 0.8564773201942444
Pérdida final del modelo: 2.3050
```

Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Redes Neuronales, también se observa los resultados de MCC, AUC y la especificidad de cada clase, para así obtener el resultado promedio de MCC, AUC y especificidad en función de las 3 clases.

- **Redes Neuronales Convolucionales**

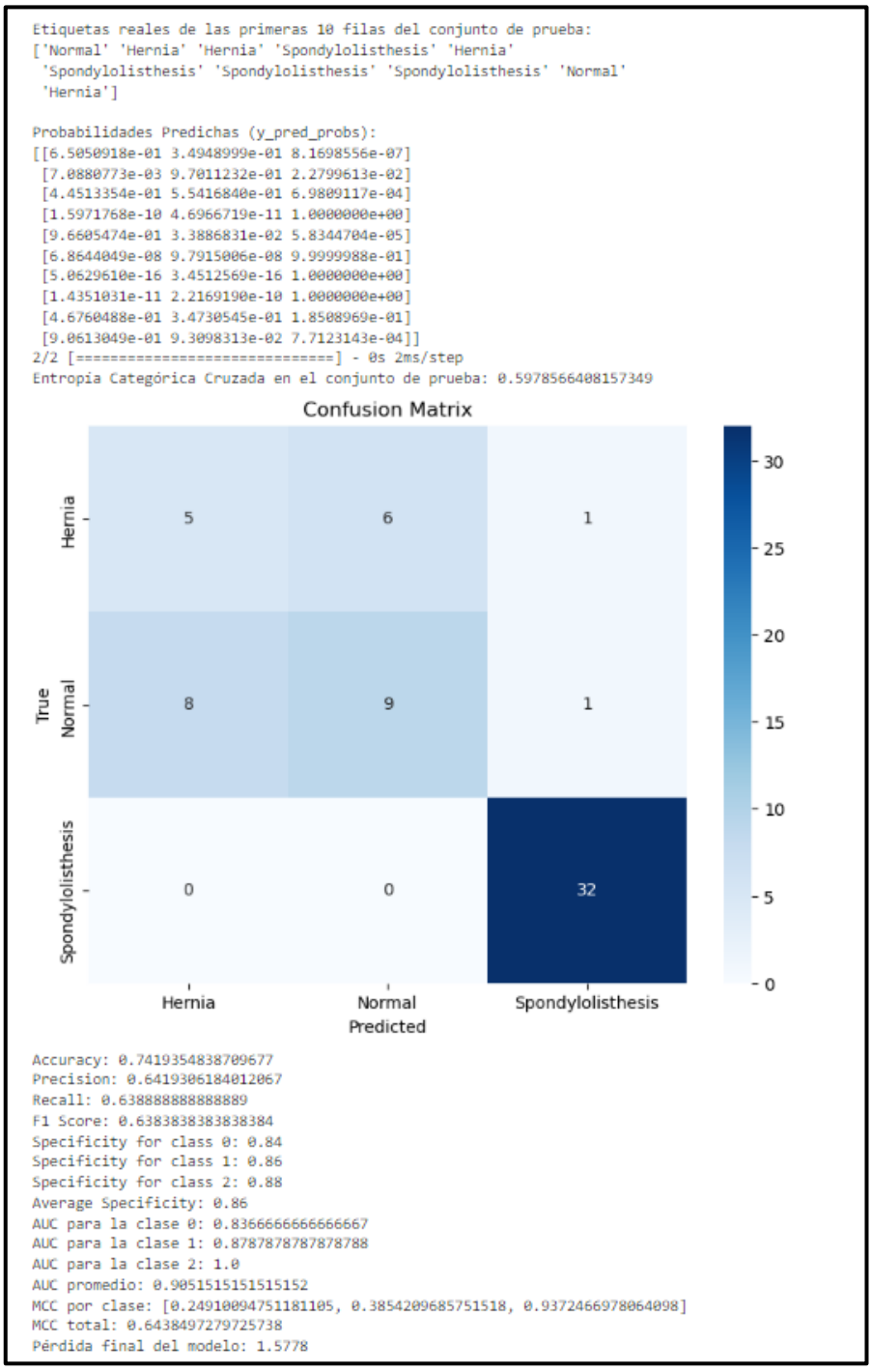
Figura Nº 145: Reporte de entrenamiento del algoritmo – RNC

```
Mejores hiperparámetros: <keras_tuner.src.engine.hyperparameters.hyperparameters.HyperParameters object at 0x000001C8CA7044D0>
Epoch 1/30
5/5 [==============================] - 1s 48ms/step - loss: 7.9163 - accuracy: 0.4240 - val_loss: 7.9748 - val_accuracy: 0.0423
Epoch 2/30
5/5 [==============================] - 0s 8ms/step - loss: 7.0685 - accuracy: 0.3640 - val_loss: 3.7816 - val_accuracy: 0.0845
Epoch 3/30
5/5 [==============================] - 0s 9ms/step - loss: 6.7477 - accuracy: 0.3887 - val_loss: 1.8803 - val_accuracy: 0.3239
Epoch 4/30
5/5 [==============================] - 0s 10ms/step - loss: 6.2742 - accuracy: 0.4064 - val_loss: 1.7172 - val_accuracy: 0.4225
Epoch 5/30
5/5 [==============================] - 0s 17ms/step - loss: 4.6601 - accuracy: 0.4700 - val_loss: 1.7336 - val_accuracy: 0.4085
Epoch 6/30
5/5 [==============================] - 0s 13ms/step - loss: 4.1949 - accuracy: 0.4982 - val_loss: 1.7910 - val_accuracy: 0.3944
Epoch 7/30
5/5 [==============================] - 0s 9ms/step - loss: 3.8581 - accuracy: 0.5583 - val_loss: 1.9389 - val_accuracy: 0.3662
Epoch 8/30
5/5 [==============================] - 0s 18ms/step - loss: 4.4021 - accuracy: 0.4876 - val_loss: 1.7427 - val_accuracy: 0.4366
Epoch 9/30
5/5 [==============================] - 0s 8ms/step - loss: 3.4503 - accuracy: 0.5936 - val_loss: 1.4088 - val_accuracy: 0.5211
Epoch 10/30
5/5 [==============================] - 0s 8ms/step - loss: 4.2462 - accuracy: 0.5088 - val_loss: 1.2204 - val_accuracy: 0.5352
Epoch 11/30
5/5 [==============================] - 0s 8ms/step - loss: 3.7412 - accuracy: 0.5548 - val_loss: 1.2086 - val_accuracy: 0.5493
Epoch 12/30
5/5 [==============================] - 0s 8ms/step - loss: 3.4892 - accuracy: 0.5442 - val_loss: 1.2037 - val_accuracy: 0.5775
Epoch 13/30
5/5 [==============================] - 0s 8ms/step - loss: 3.5785 - accuracy: 0.5406 - val_loss: 1.1757 - val_accuracy: 0.6056
Epoch 14/30
5/5 [==============================] - 0s 15ms/step - loss: 2.6371 - accuracy: 0.6007 - val_loss: 1.2003 - val_accuracy: 0.5211
Epoch 15/30
5/5 [==============================] - 0s 15ms/step - loss: 2.5861 - accuracy: 0.6113 - val_loss: 1.2306 - val_accuracy: 0.4648
Epoch 16/30
5/5 [==============================] - 0s 7ms/step - loss: 2.8473 - accuracy: 0.5901 - val_loss: 1.1523 - val_accuracy: 0.6197
Epoch 17/30
5/5 [==============================] - 0s 8ms/step - loss: 2.5441 - accuracy: 0.6148 - val_loss: 1.1140 - val_accuracy: 0.6338
Epoch 18/30
5/5 [==============================] - 0s 7ms/step - loss: 2.4309 - accuracy: 0.6184 - val_loss: 1.0537 - val_accuracy: 0.6620
Epoch 19/30
5/5 [==============================] - 0s 7ms/step - loss: 2.8368 - accuracy: 0.6078 - val_loss: 1.0595 - val_accuracy: 0.6479
Epoch 20/30
5/5 [==============================] - 0s 7ms/step - loss: 2.4628 - accuracy: 0.6113 - val_loss: 1.1018 - val_accuracy: 0.6338
Epoch 21/30
5/5 [==============================] - 0s 8ms/step - loss: 1.9928 - accuracy: 0.6466 - val_loss: 1.1273 - val_accuracy: 0.6056
Epoch 22/30
5/5 [==============================] - 0s 10ms/step - loss: 2.5365 - accuracy: 0.6219 - val_loss: 1.1505 - val_accuracy: 0.5775
Epoch 23/30
5/5 [==============================] - 0s 10ms/step - loss: 2.5191 - accuracy: 0.6537 - val_loss: 1.1121 - val_accuracy: 0.6479
Epoch 24/30
5/5 [==============================] - 0s 15ms/step - loss: 2.4175 - accuracy: 0.6113 - val_loss: 1.0758 - val_accuracy: 0.6620
Epoch 25/30
5/5 [==============================] - 0s 8ms/step - loss: 2.4001 - accuracy: 0.6219 - val_loss: 1.0248 - val_accuracy: 0.6479
Epoch 26/30
5/5 [==============================] - 0s 11ms/step - loss: 2.3372 - accuracy: 0.6466 - val_loss: 0.9969 - val_accuracy: 0.6761
Epoch 27/30
5/5 [==============================] - 0s 8ms/step - loss: 1.8990 - accuracy: 0.6643 - val_loss: 1.0628 - val_accuracy: 0.6479
Epoch 28/30
5/5 [==============================] - 0s 9ms/step - loss: 2.3805 - accuracy: 0.6360 - val_loss: 1.1167 - val_accuracy: 0.6620
Epoch 29/30
5/5 [==============================] - 0s 14ms/step - loss: 1.7992 - accuracy: 0.6820 - val_loss: 1.1865 - val_accuracy: 0.6338
Epoch 30/30
5/5 [==============================] - 0s 9ms/step - loss: 1.5778 - accuracy: 0.6961 - val_loss: 1.1853 - val_accuracy: 0.6338
2/2 [==============================] - 0s 5ms/step
```

Fuente: Elaboración propia

Se visualiza los mejores hiperparámetros y el entrenamiento del modelo de redes neuronales, aunque cuando el modelo es entrenado nuevamente, los resultados cambian. Por lo tanto, la figura Nº 144 representa a uno de los entrenamientos realizado en el modelo de redes neuronales.

Figura Nº 146: Reporte de resultados del algoritmo – RNC

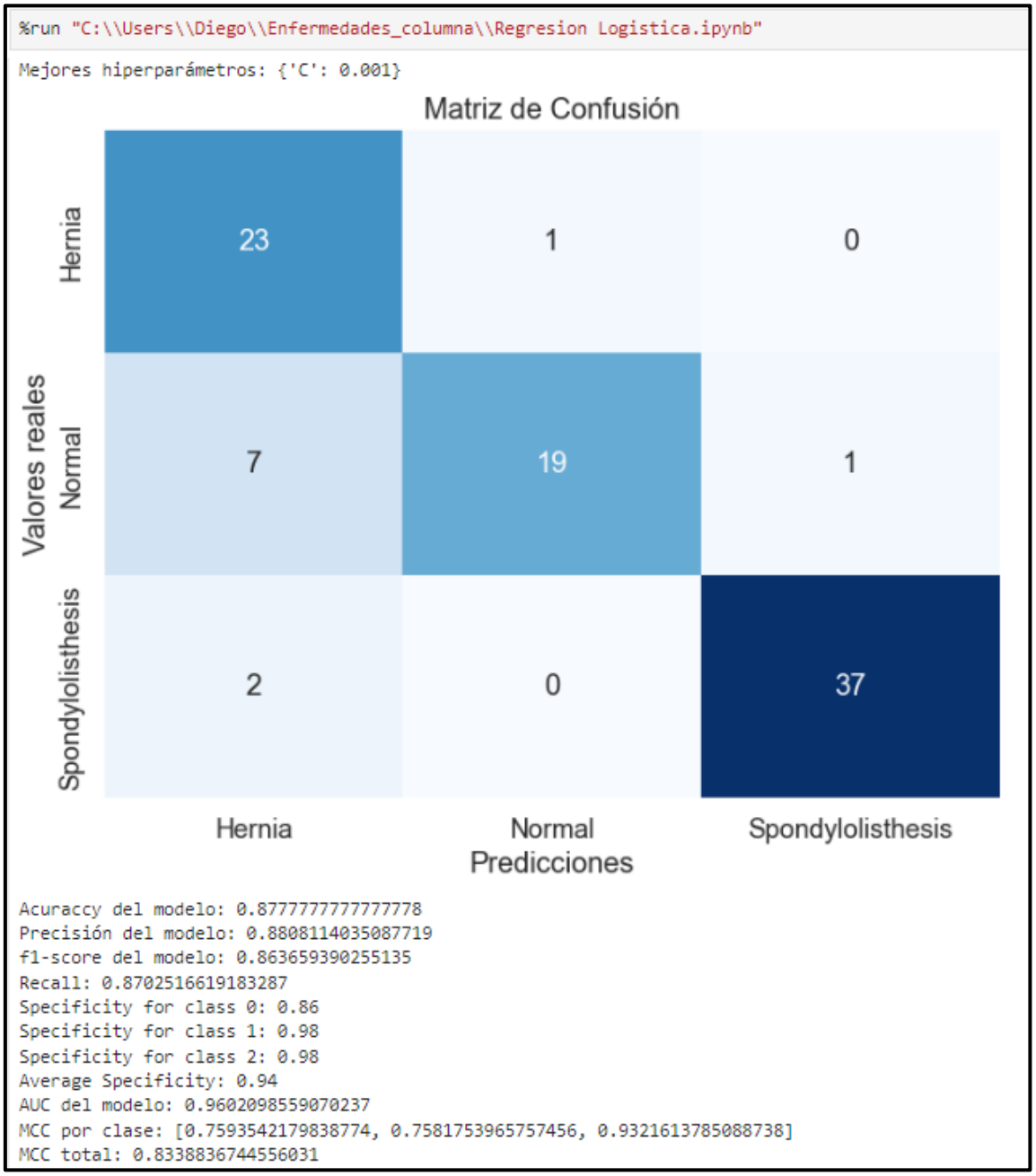


Fuente: Elaboración propia

Se visualiza el resultado de las etiquetas reales y probabilidades, la matriz de confusión y el resultado de todas las métricas para el algoritmo Redes Neuronales Convolucionales, también se observa los resultados de MCC, AUC y la especificidad de cada clase, para así obtener el resultado promedio de MCC, AUC y especificidad en función de las 3 clases.

- **Regresión Logística**

Figura Nº 147: Reporte de resultados del algoritmo – RL

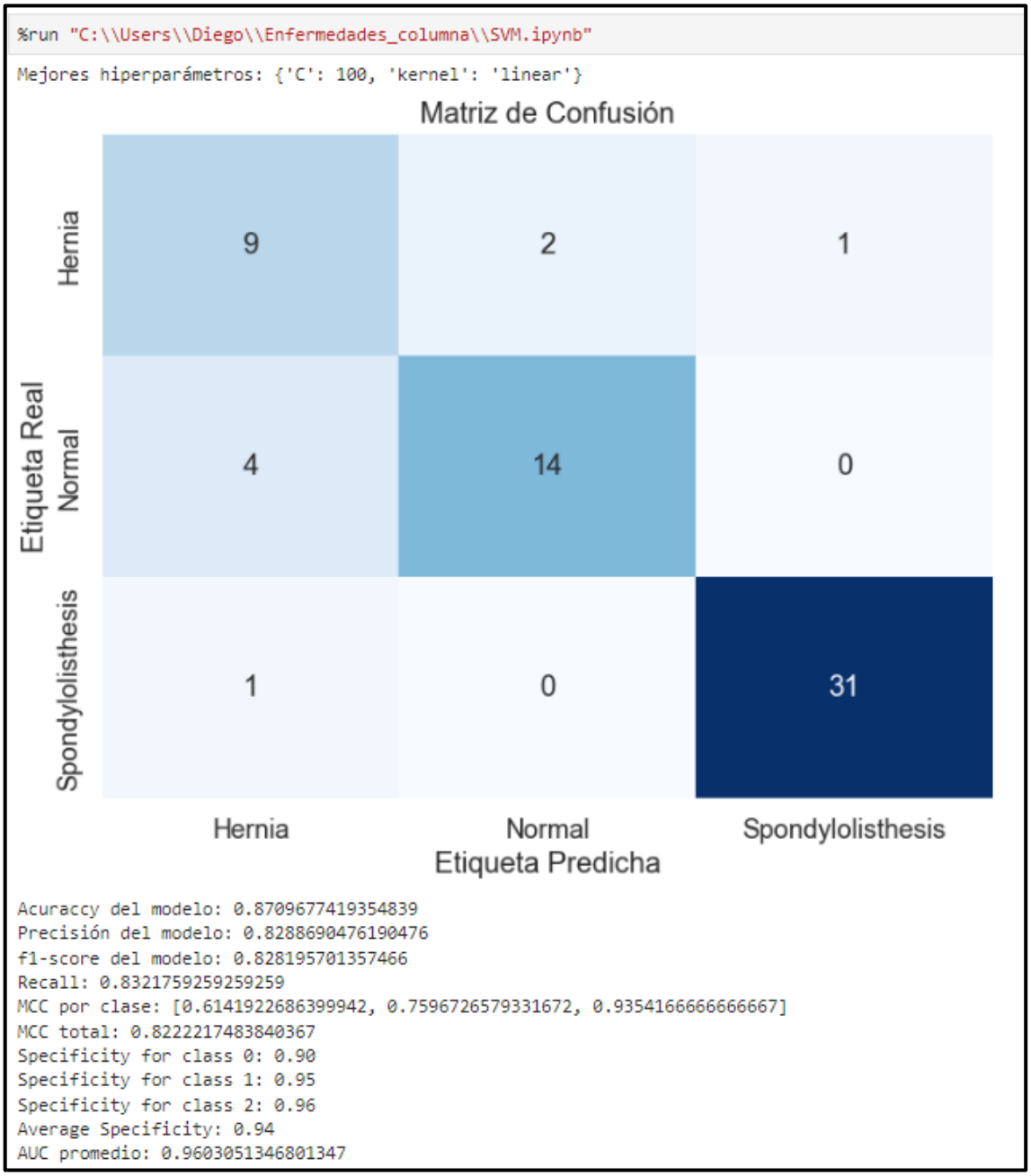


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Regresión Logística, también se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Support Vector Machine**

Figura Nº 148: Reporte de resultados del algoritmo – SVM

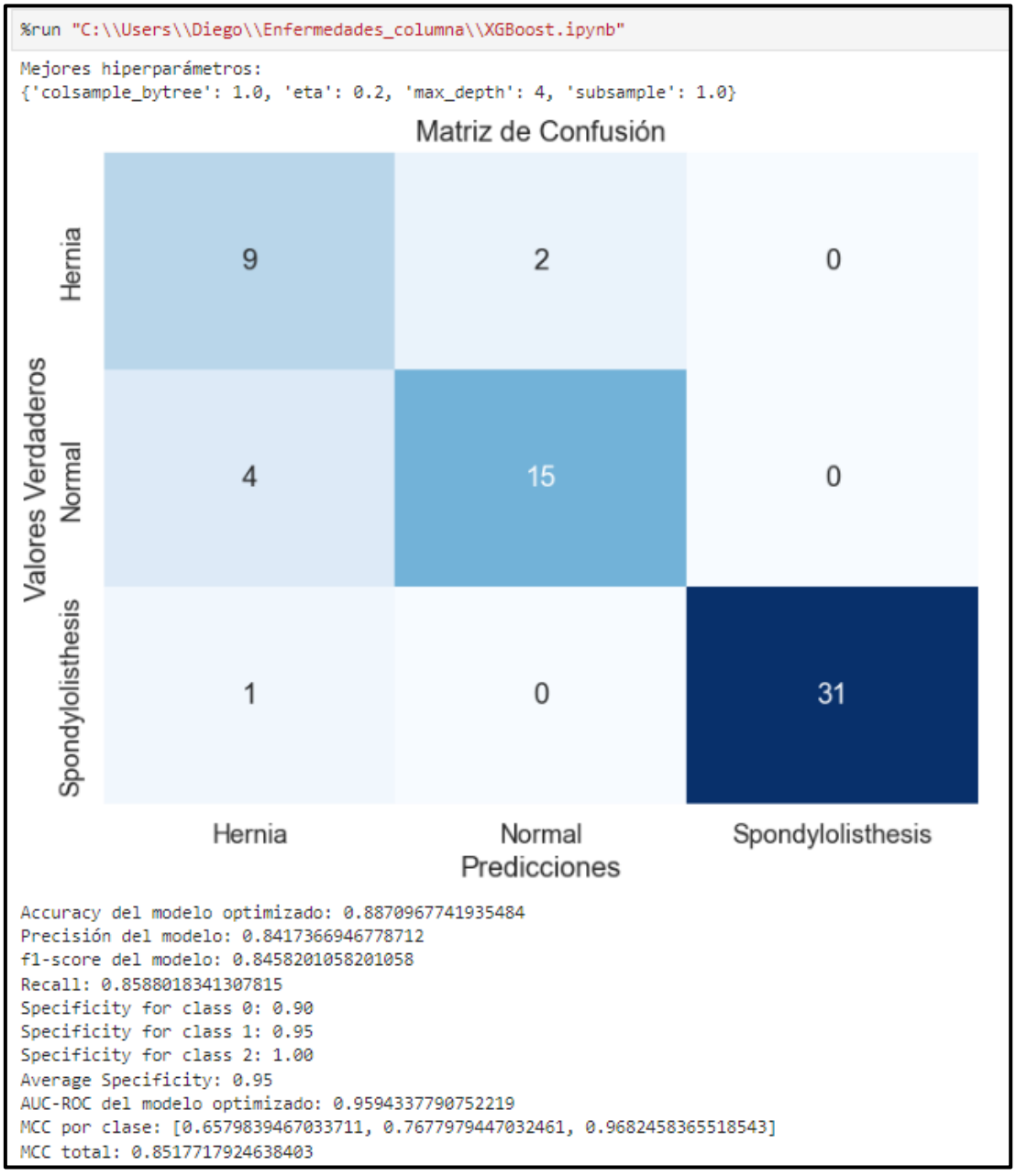


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo Support Vector Machine también se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **XGBoost**

Figura Nº 149: Reporte de resultados del algoritmo – XGB

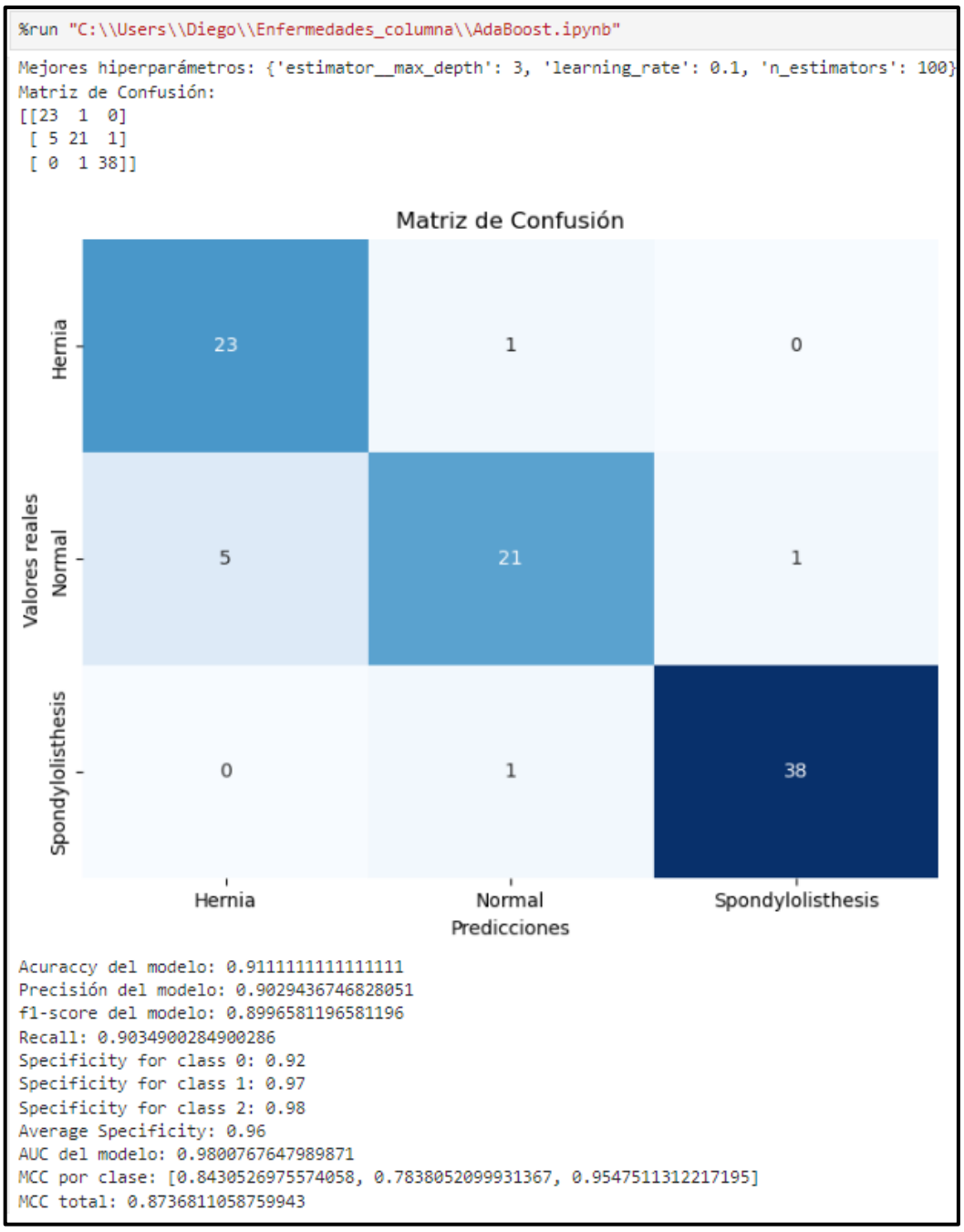


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo XGBoost se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **AdaBoost**

Figura Nº 150: Reporte de resultados del algoritmo – ADA

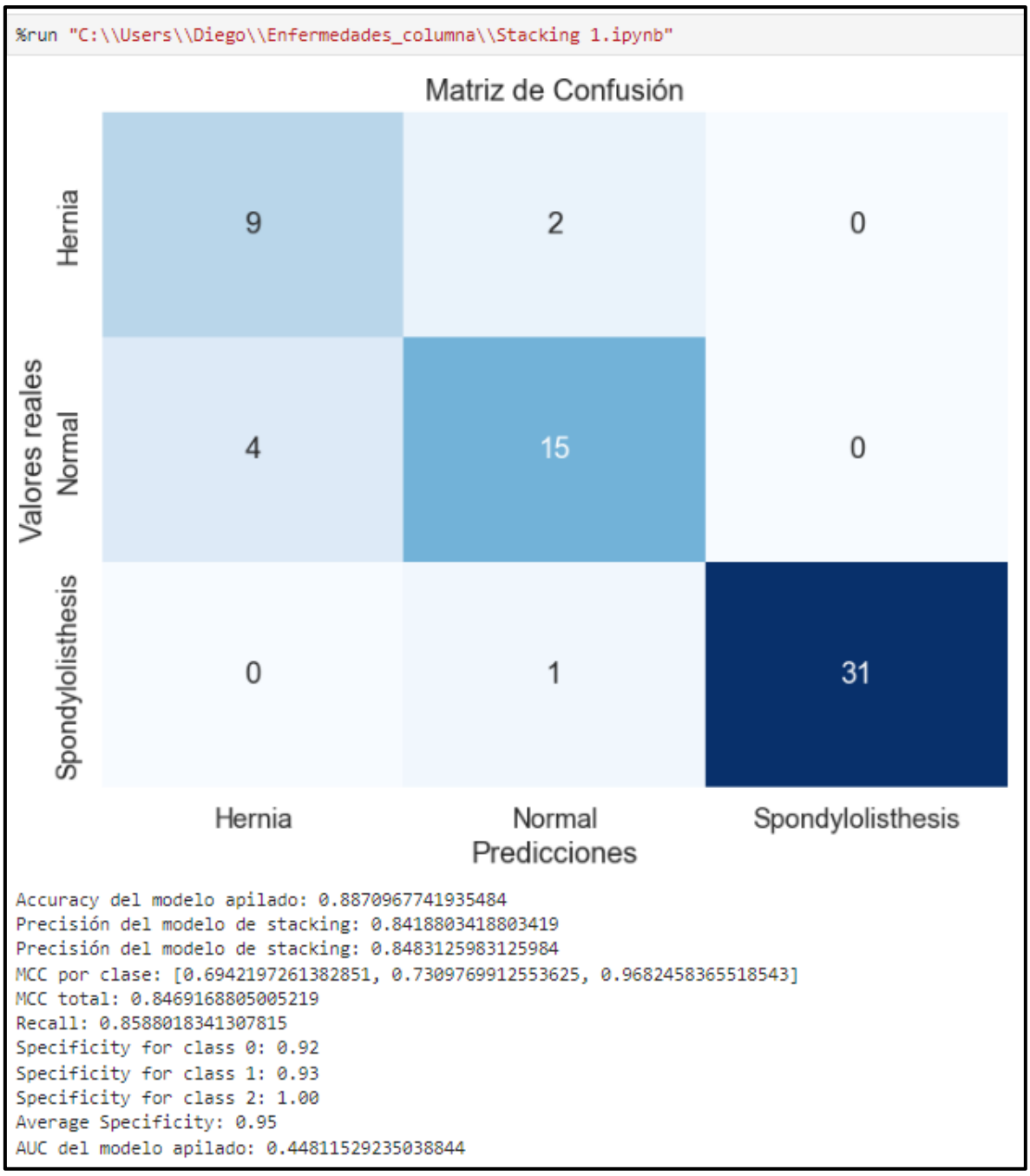


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo XGBoost se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Stacking 1**

Figura Nº 151: Reporte de resultados del algoritmo – Stacking 1

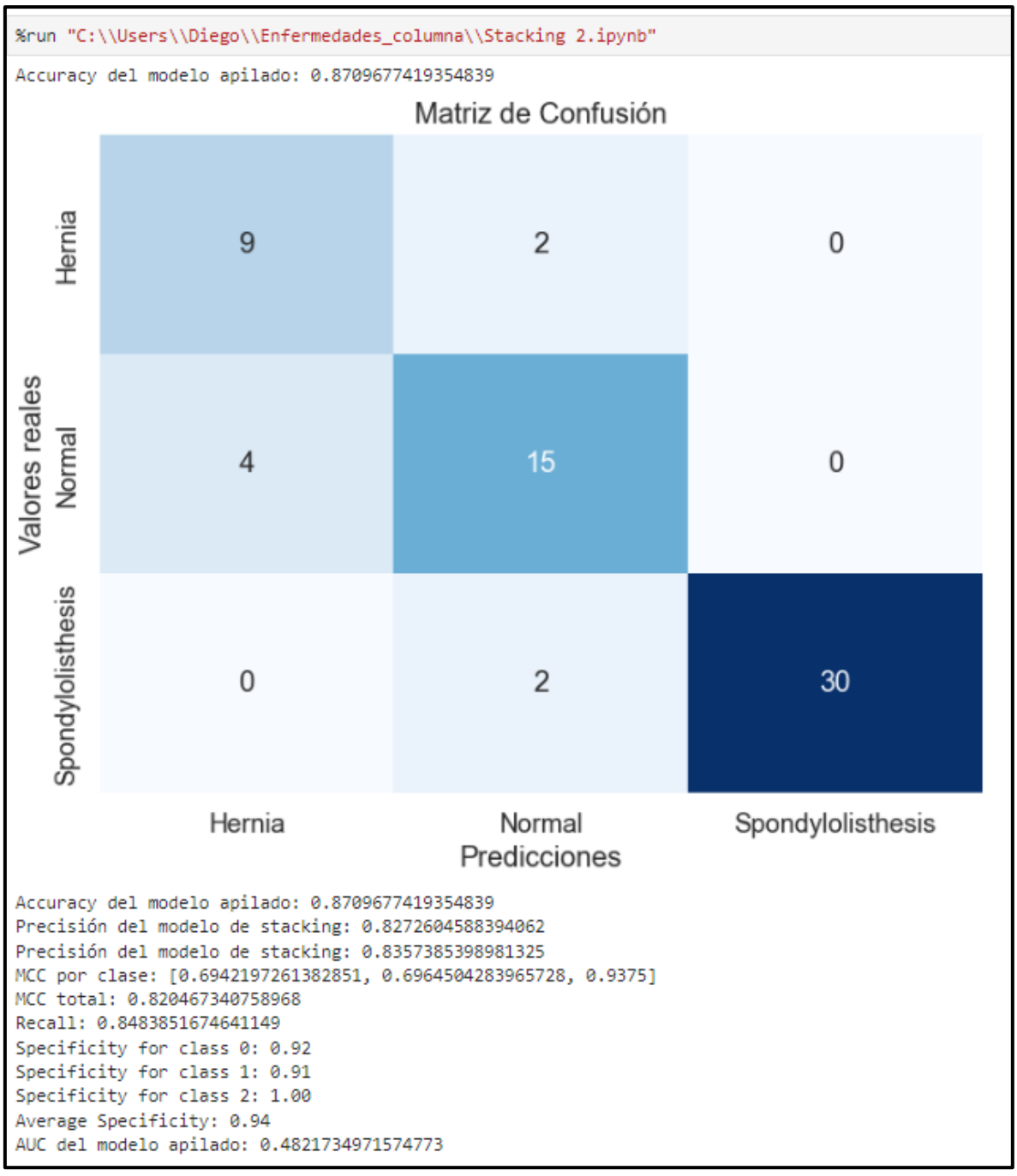


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo XGBoost se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Stacking 2**

Figura Nº 152: Reporte de resultados del algoritmo – Stacking 2

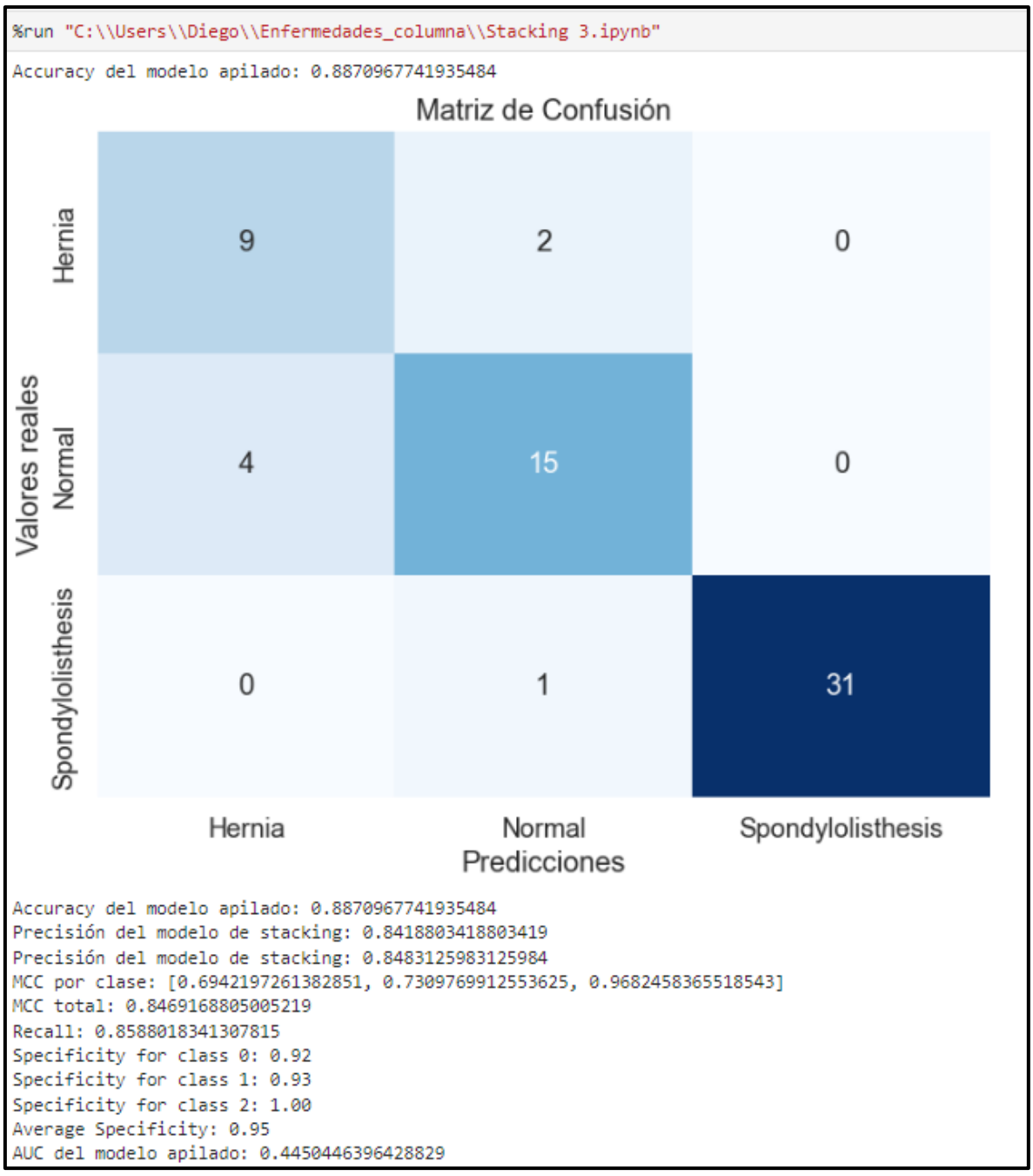


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo XGBoost se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Stacking 3**

Figura Nº 153: Reporte de resultados del algoritmo – Stacking 3

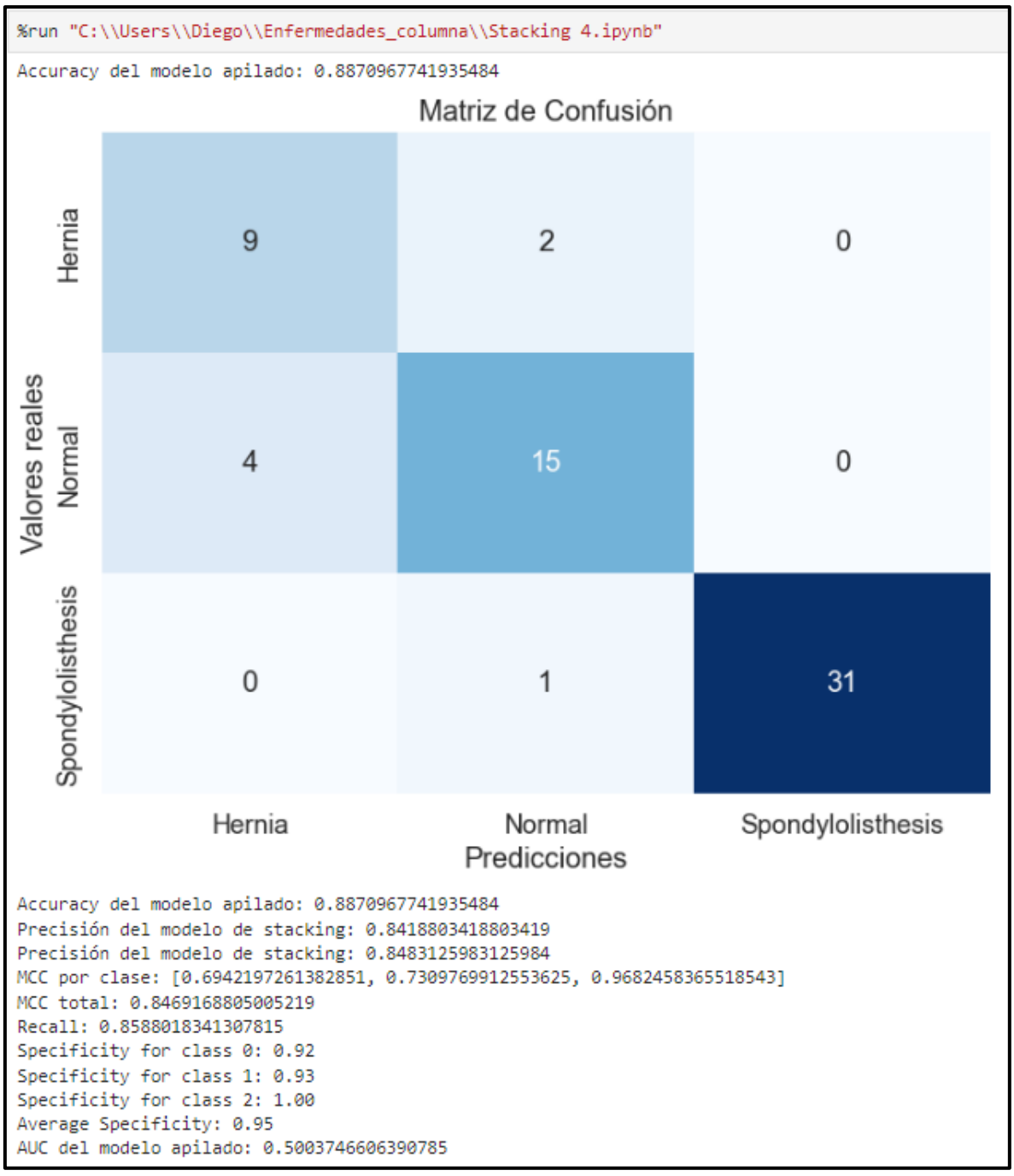


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo XGBoost se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Stacking 4**

Figura Nº 154: Reporte de resultados del algoritmo – Stacking 4

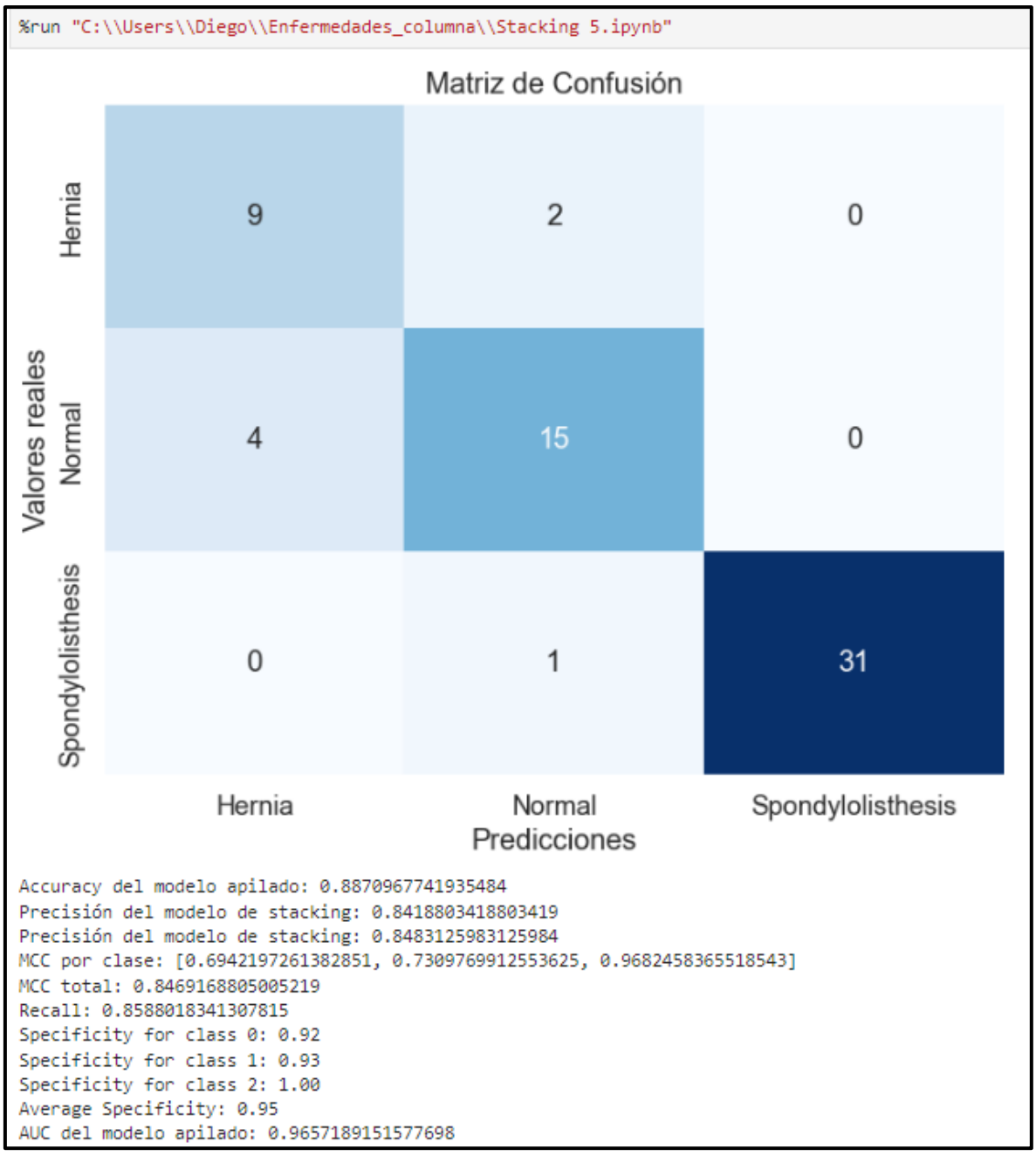


Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo XGBoost se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Stacking 5**

Figura Nº 155: Reporte de resultados del algoritmo – Stacking 5



Fuente: Elaboración propia

Se visualiza los resultados de los mejores hiperparámetros, el resultado de la matriz de confusión y el resultado de todas las métricas para el algoritmo XGBoost se observa los resultados de MCC y la especificidad de cada clase, para así obtener el resultado promedio de MCC y especificidad en función de las 3 clases.

- **Sistema inteligente con Machine Learning**

Figura Nº 156: Librerías usadas para el Sistema Inteligente

```
import pickle
import pandas as pd
import numpy as np
import streamlit as st
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from streamlit_option_menu import option_menu
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix


from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

Fuente: Elaboración propia

Se importan las librerías “streamlit” y “streamlit_option” para el diseño del sistema, para la conexión de los modelos se usó “pickle”, “sklearm.model_selection”, “sklearn.metrics” para las métricas y los algoritmos de aprendizaje automático.

Figura Nº 157: Cargar los modelos

```
# Cargar modelos
adb_model = pickle.load(open('D:/pro_columna/ada_model.sav', 'rb'))
randomf_model = pickle.load(open('D:/pro_columna/randomf_model.sav', 'rb'))
stack1_model = pickle.load(open('D:/pro_columna/stack1_model.sav', 'rb'))

```

Fuente: Elaboración propia

Se asigno las variables adb_model, randomf_model y stack1_model para cargar cada modelo que fue guardado.

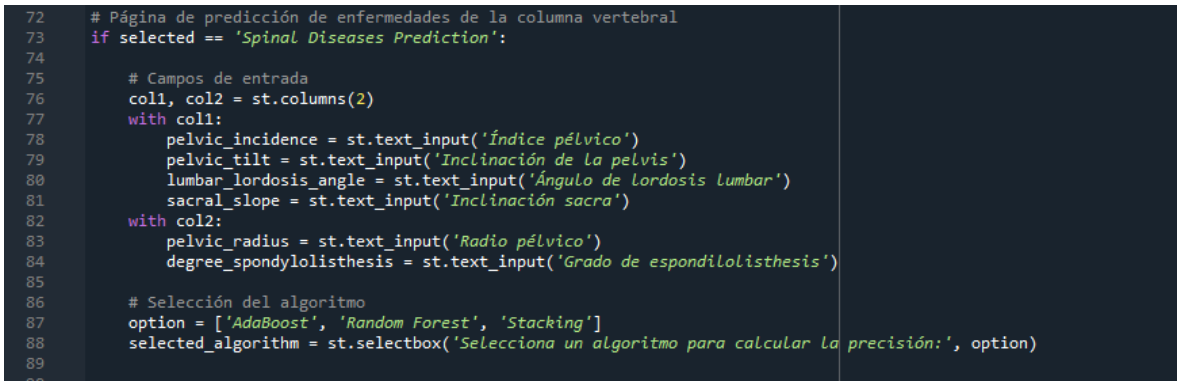
Figura Nº 158: Menú de navegación del sistema inteligente

```
# sidebar for navigation
with st.sidebar:

    with st.sidebar:
        selected = option_menu("Main Menu", ['Inicio','Spinal Diseases Prediction', 'Subir datos'],
            icons=['house', 'pencil-square', 'cloud-upload'], menu_icon="cast", default_index=1)
        selected
```

Fuente: Elaboración propia

Se menciona los apartados por los que está compuesto el menú de navegación.

Figura Nº 159: Campos y selección del algoritmo



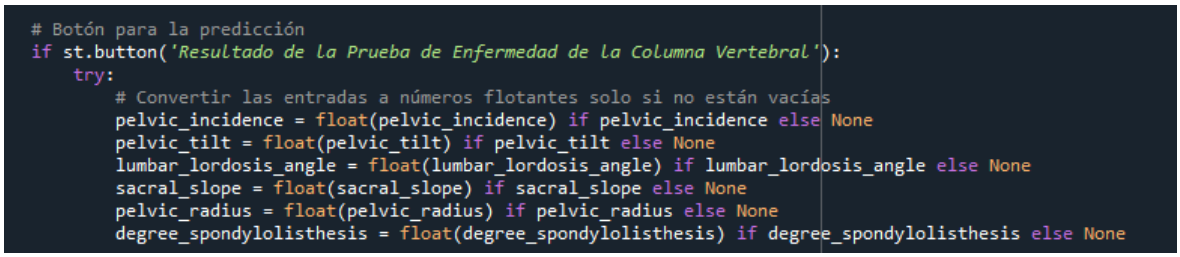
```
# Página de predicción de enfermedades de la columna vertebral
if selected == 'Spinal Diseases Prediction':

    # Campos de entrada
    col1, col2 = st.columns(2)
    with col1:
        pelvic_incidence = st.text_input('Índice pélvico')
        pelvic_tilt = st.text_input('Inclinación de la pelvis')
        lumbar_lordosis_angle = st.text_input('Ángulo de lordosis lumbar')
        sacral_slope = st.text_input('Inclinación sacra')
    with col2:
        pelvic_radius = st.text_input('Radio pélvico')
        degree_spondylolisthesis = st.text_input('Grado de espondilolisthesis')

    # Selección del algoritmo
    option = ['AdaBoost', 'Random Forest', 'Stacking']
    selected_algorithm = st.selectbox('Selecciona un algoritmo para calcular la precisión:', option)
```

Fuente: Elaboración propia

Se crean los campos a llenar y se les asigna una variable en función a las variables con las que cuenta la BD, luego se visualiza el selectbox para la selección del algoritmo.

Continuando con el código, se creó el botón para realizar la predicción

Figura Nº 160: Botón para realizar la predicción

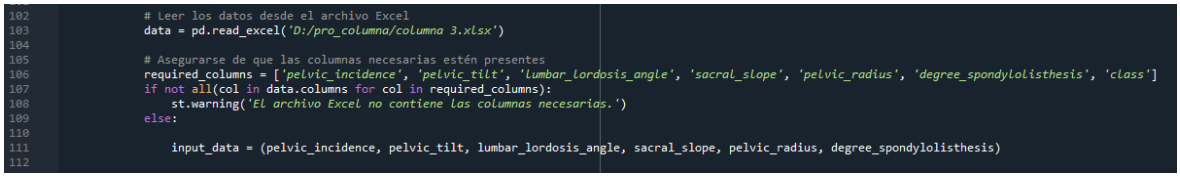


```
# Botón para la predicción
if st.button('Resultado de la Prueba de Enfermedad de la Columna Vertebral'):
    try:
        # Convertir las entradas a números flotantes solo si no están vacías
        pelvic_incidence = float(pelvic_incidence) if pelvic_incidence else None
        pelvic_tilt = float(pelvic_tilt) if pelvic_tilt else None
        lumbar_lordosis_angle = float(lumbar_lordosis_angle) if lumbar_lordosis_angle else None
        sacral_slope = float(sacral_slope) if sacral_slope else None
        pelvic_radius = float(pelvic_radius) if pelvic_radius else None
        degree_spondylolisthesis = float(degree_spondylolisthesis) if degree_spondylolisthesis else None
```

Fuente: Elaboración propia

Se creo el botón llamado Resultado de la Prueba de Enfermedad de la columna vertebral. Luego se hizo la conversión de los datos de tipo numérico a números de tipo flotante.

Figura Nº 161: Columnas necesarias a llenar



```
            # Leer los datos desde el archivo Excel
            data = pd.read_excel('D:/pro_columna/columna 3.xlsx')

            # Asegurarse de que las columnas necesarias estén presentes
            required_columns = ['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class']
            if not all(col in data.columns for col in required_columns):
                st.warning('El archivo Excel no contiene las columnas necesarias.')
            else:

                input_data = (pelvic_incidence, pelvic_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius, degree_spondylolisthesis)
```

Fuente: Elaboración propia

Se lee el archivo Excel que contiene los datos con los que trabajara la predicción y se creó la variable required_columns, para asegurarse que las variables sean las mismas con las que cuenta la BD.

Figura Nº 162: Funcionamiento del botón de predicción

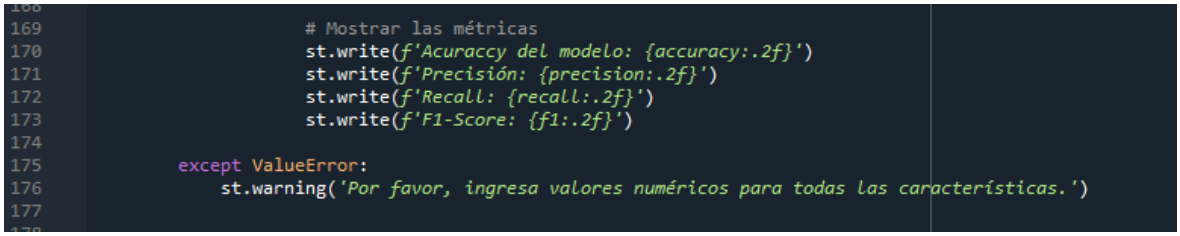
```
                # Seleccionar las columnas necesarias
                X = data[['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis']].values
                Y = data['class']

                # Dividir los datos en conjuntos de entrenamiento y prueba
                X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

                # Seleccionar el modelo según la opción elegida
                if selected_algorithm == 'AdaBoost':
                   model = adb_model  # Ajusta el modelo según tus necesidades
                elif selected_algorithm == 'Random Forest':
                   model = randomf_model  # Ajusta el modelo según tus necesidades
                elif selected_algorithm == 'Stacking':
                   model = stack1_model  # Ajusta el modelo según tus necesidades

                # Verificar si el modelo es válido
                if model is None:
                    st.warning('Por favor, selecciona un modelo válido.')
                else:
                    # Entrenar el modelo
                    model.fit(X_train, y_train)

                    #Realizar predicciones en el conjunto de prueba
                    y_pred = model.predict(X_test)

                    # Cambiar el input_data a un array de numpy
                    input_data_as_numpy_array = np.asarray(input_data)

                    # Reorganizar el array ya que estamos haciendo la predicción para una instancia
                    input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)

                    # Obtener las probabilidades de pertenencia a cada clase
                    prediction_probabilities = model.predict_proba(input_data_reshaped)

                    # Obtener la clase con la probabilidad más alta
                    predicted_class = np.argmax(prediction_probabilities)

                    # Mapear el índice de clase predicho a la etiqueta de clase real
                    class_labels = {0: 'Hernia', 1: 'Normal', 2: 'Espondilolistesis'}
                    predicted_class_label = class_labels[predicted_class]

                    # Mostrar el mensaje correspondiente a la clase predicha
                    if predicted_class_label == 'Hernia':
                        st.write('La persona tiene Hernia en la columna vertebral.')
                    elif predicted_class_label == 'Espondilolistesis':
                        st.write('La persona tiene Espondilolistesis en la columna vertebral.')
                    elif predicted_class_label == 'Normal':
                        st.write('La persona no tiene ninguna enfermedad de la columna vertebral.')

                    # Calcular las métricas
                    accuracy = accuracy_score(y_test, y_pred)
                    precision = precision_score(y_test, y_pred, average='macro')
                    recall = recall_score(y_test, y_pred, average='macro')
                    f1 = f1_score(y_test, y_pred, average='macro')

```

Fuente: Elaboración propia

Se visualiza los de X con los que se trabajará al llenar los datos y la variable Y (class), la cual es la variable objetivo para realizar la predicción. Se procedió a dividir la data. Se realiza la selección del algoritmo, para así proceder a entrenar el modelo en donde se obtendrá como resultado que, si la persona cumple con los datos que pertenecen a la categoría Hernia de la variable class, se visualizara el mensaje “La persona tiene Hernia en la columna vertebral”, Si la persona cumple con los datos que pertenecen a la categoría Spondylolisthesis de la variable class, se visualizara el mensaje “La persona tiene Espondilolistesis en la columna vertebral” y si la persona no cumple con ninguna de esas 2 categorías, existe la tercera categoría Normal con la cuenta la variable class de la BD y permite conocer si la persona no tiene ninguna enfermedad de la columna vertebral.

En función a los resultados del algoritmo se calcula las métricas para poder evaluar el rendimiento del modelo.

Figura Nº 163: Imprimir datos de las métricas

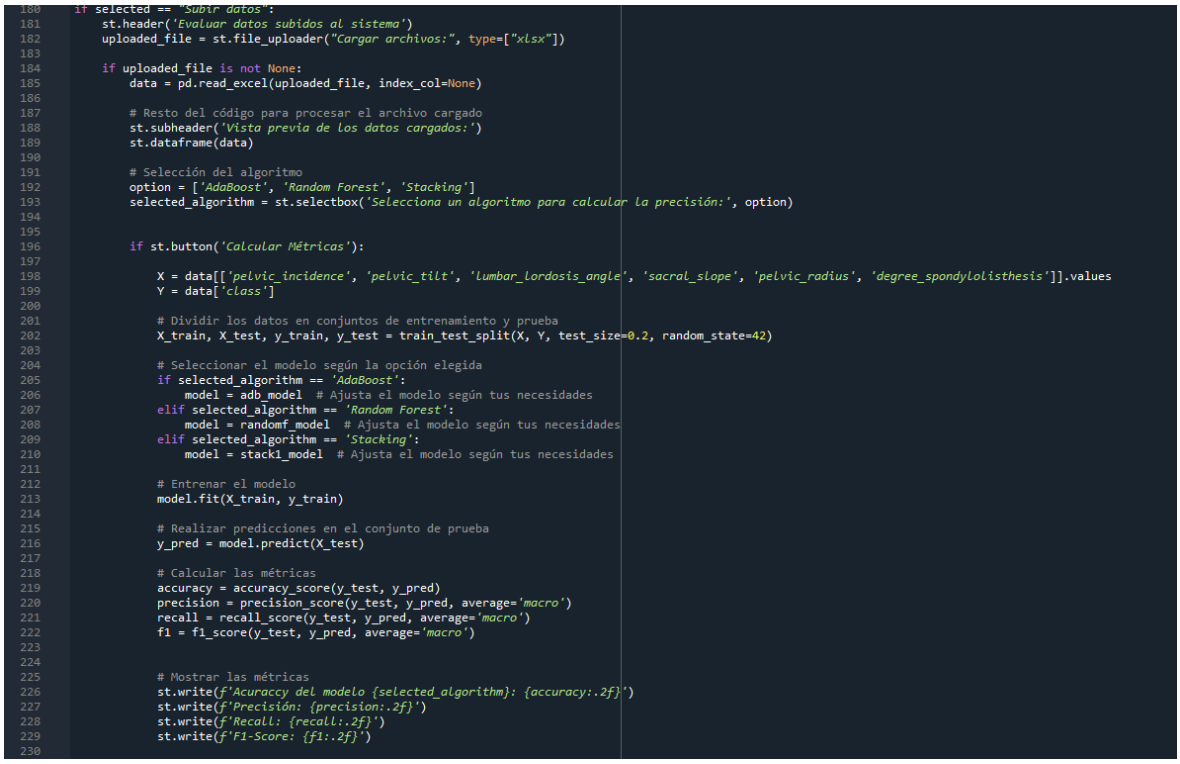


```
                    # Mostrar las métricas
                    st.write(f'Acuraccy del modelo: {accuracy:.2f}')
                    st.write(f'Precisión: {precision:.2f}')
                    st.write(f'Recall: {recall:.2f}')
                    st.write(f'F1-Score: {f1:.2f}')

            except ValueError:
                st.warning('Por favor, ingresa valores numéricos para todas las características.')
```

Fuente: Elaboración propia

Se muestran los resultados de cada métrica y se ejecutará el mensaje de error “Por favor, ingresa valores numéricos para todas las características” si no se cumple con el llenado correcto de los datos a evaluar.

Figura Nº 164: Opción Subir datos



```
if selected == "Subir datos":
    st.header('Evaluar datos subidos al sistema')
    uploaded_file = st.file_uploader("Cargar archivos:", type=["xlsx"])

    if uploaded_file is not None:
        data = pd.read_excel(uploaded_file, index_col=None)

        # Resto del código para procesar el archivo cargado
        st.subheader('Vista previa de los datos cargados:')
        st.dataframe(data)

        # Selección del algoritmo
        option = ['AdaBoost', 'Random Forest', 'Stacking']
        selected_algorithm = st.selectbox('Selecciona un algoritmo para calcular la precisión:', option)


        if st.button('Calcular Métricas'):

            X = data[['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis']].values
            Y = data['class']

            # Dividir los datos en conjuntos de entrenamiento y prueba
            X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

            # Seleccionar el modelo según la opción elegida
            if selected_algorithm == 'AdaBoost':
                model = adb_model  # Ajusta el modelo según tus necesidades
            elif selected_algorithm == 'Random Forest':
                model = randomf_model  # Ajusta el modelo según tus necesidades
            elif selected_algorithm == 'Stacking':
                model = stack1_model  # Ajusta el modelo según tus necesidades

            # Entrenar el modelo
            model.fit(X_train, y_train)

            # Realizar predicciones en el conjunto de prueba
            y_pred = model.predict(X_test)

            # Calcular las métricas
            accuracy = accuracy_score(y_test, y_pred)
            precision = precision_score(y_test, y_pred, average='macro')
            recall = recall_score(y_test, y_pred, average='macro')
            f1 = f1_score(y_test, y_pred, average='macro')


            # Mostrar las métricas
            st.write(f'Acuraccy del modelo {selected_algorithm}: {accuracy:.2f}')
            st.write(f'Precisión: {precision:.2f}')
            st.write(f'Recall: {recall:.2f}')
            st.write(f'F1-Score: {f1:.2f}')
```

Fuente: Elaboración propia

Se visualiza el botón “Evaluar datos subidos al sistema”, el cual aparecerá luego de haberse subido los datos de un archivo de formato xlsx.

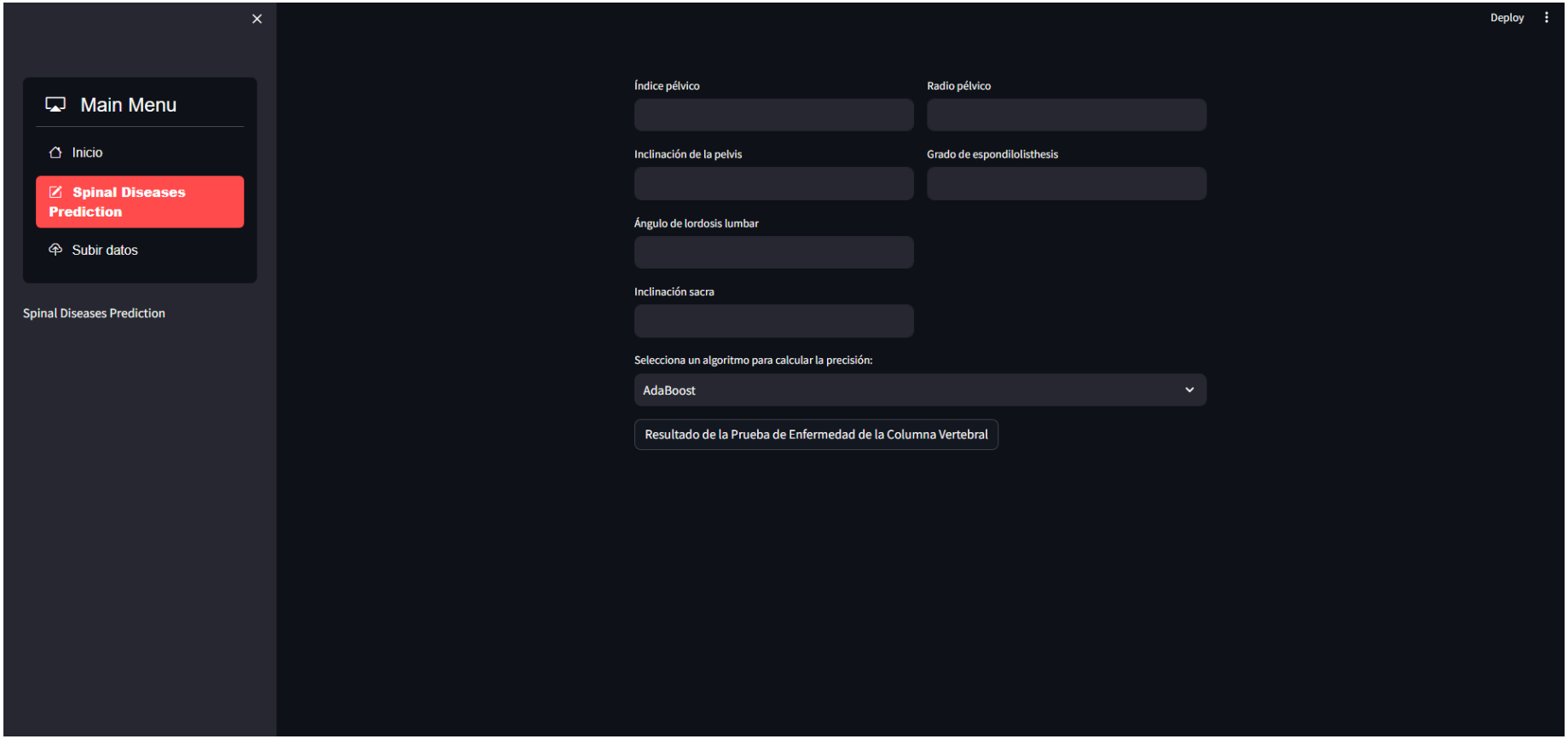
Se realiza el mismo procedimiento que se hizo en el apartado para llenar los datos, en donde se asegura que la data cuente con las variables adecuadas para realizar la evaluación en función a los algoritmos propuestos. Y como resultado se mostrará la evaluación en función de las métricas de exactitud, precisión, recall y f1-score.

Figura Nº 165: Inicio del sistema Inteligente

Fuente: Elaboración propia

En la opción de inicio se da a conocer sobre la problemática de enfermedades de la columna vertebral

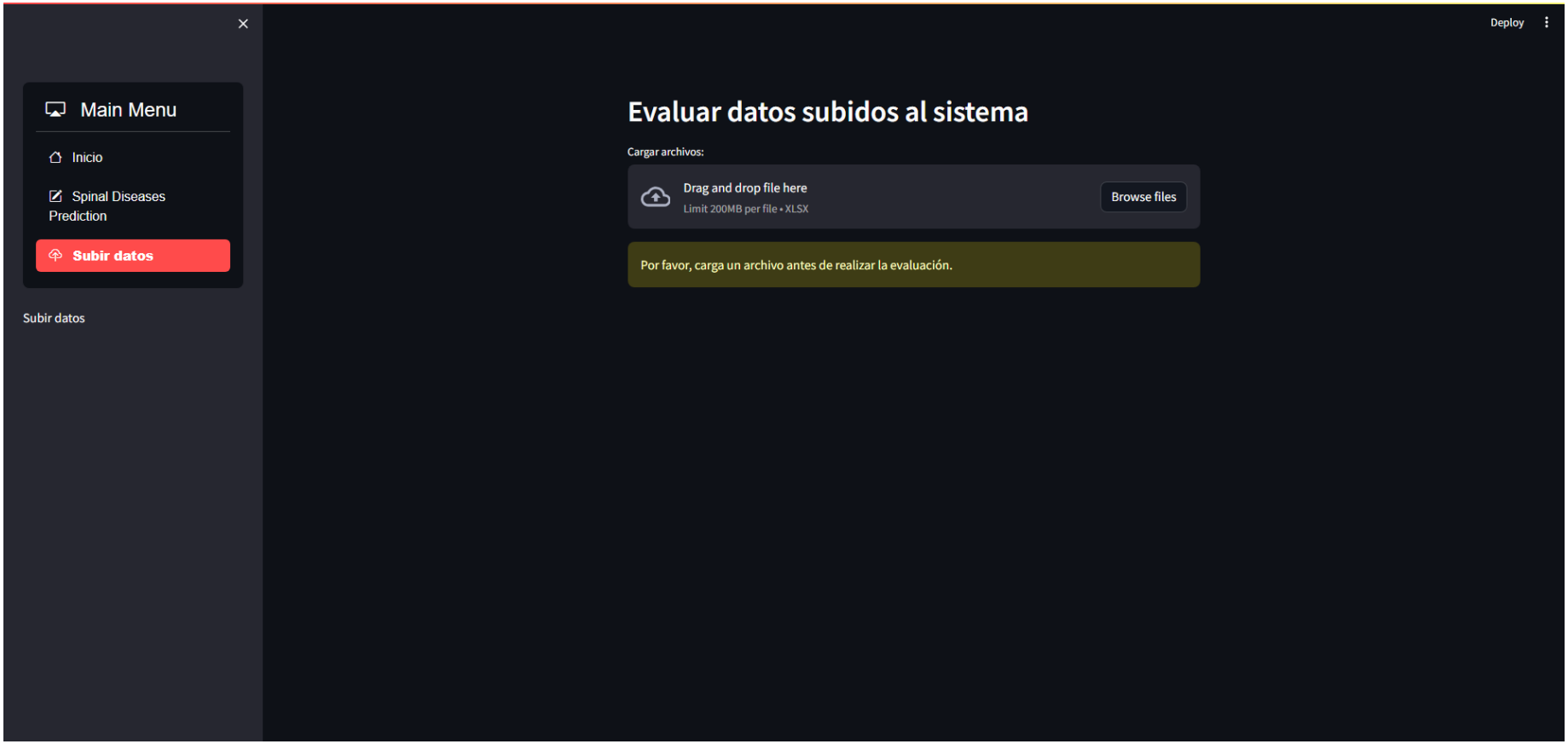
Figura Nº 166: Datos de entrada del Sistema Inteligente



Fuente: Elaboración propia

En la opción Spinal Diseases Prediction se podrá digitar los datos para realizar la predicción y se encuentra el botón “Resultado de la prueba de enfermedad de la columna vertebral” que permite obtener los resultados.

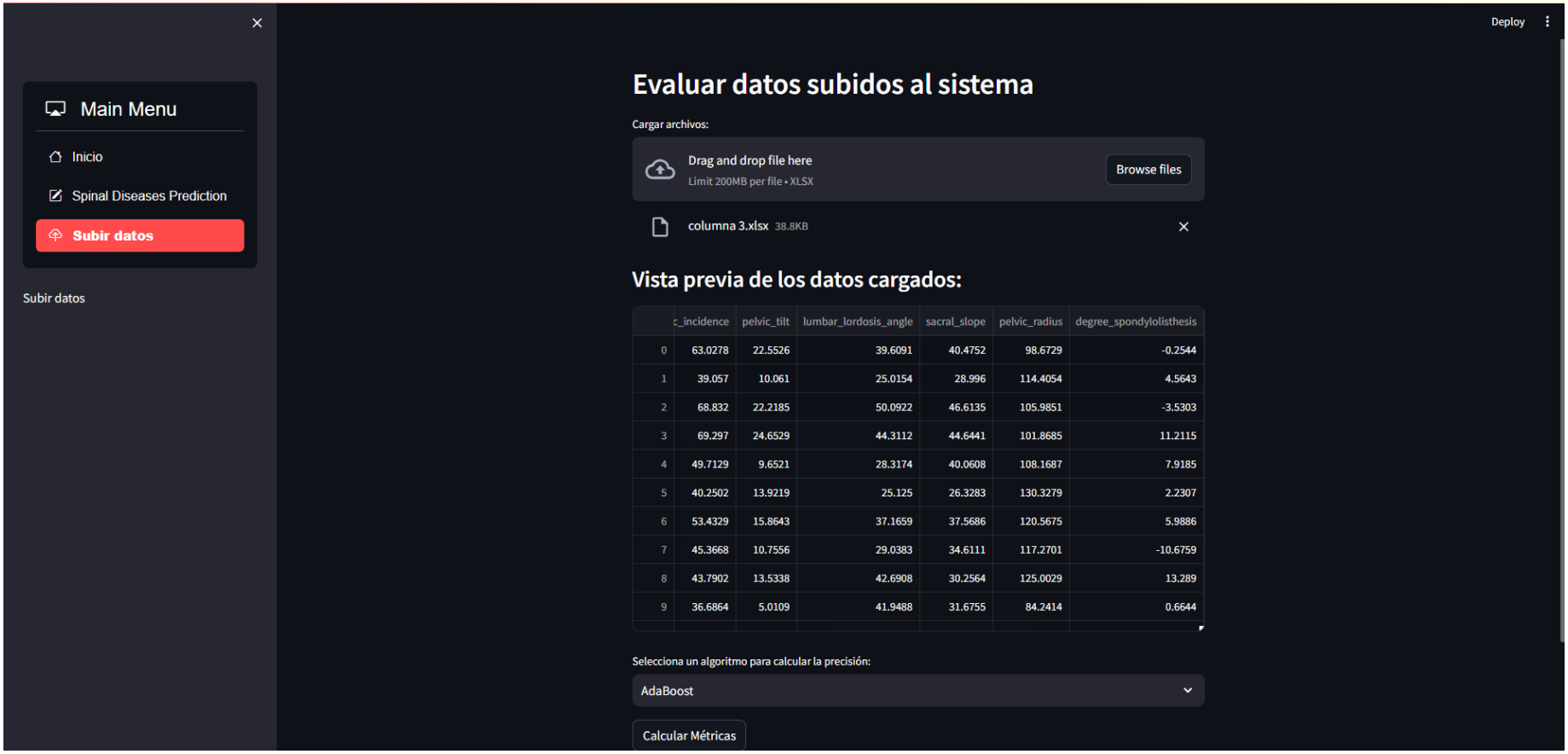
Figura Nº 167: Subir datos al Sistema Inteligente



Fuente: Elaboración propia

En la opción subir datos, se podrá cargar archivos para poder realizar la evaluación en función a las métricas y el algoritmo que sea seleccionado.

Figura Nº 168: Datos subidos al sistema



Fuente: Elaboración propia

Se visualiza una vista previa de los datos cargados y se muestra la opción para elegir el algoritmo y así poder evaluar la data subida al sistema inteligente.