UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO
CARRERA DE BIOMEDICINA

DESARROLLO DE UN MODELO DE CLASIFICACIÓN AUTOMÁTICO EN EL SOFTWARE R PARA LAS ETAPAS DE LA ENFERMEDAD RENAL CRÓNICA MEDIANTE MACHINE LEARNING

Trabajo de titulación previo a la obtención del título de:
INGENIERO BIOMÉDICO

AUTOR: ALEX FERNANDO TALLANA CASTRO

TUTOR: ING. LUIS GEOVANNY ROMERO MEJÍA

Quito - Ecuador

2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Alex Fernando Tallana Castro con documento de identificación N° 1751426444 manifiesto que:

Soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 29 de Febrero del año 2024

Atentamente,

Alex Fernando Tallana Castro
1751426444

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Alex Fernando Tallana Castro con documento de identificación No. 1751426444, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Trabajo experimental: **"DESARROLLO DE UN MODELO DE CLASIFICACIÓN AUTOMÁTICO EN EL SOFTWARE R PARA LAS ETAPAS DE LA ENFERMEDAD RENAL CRÓNICA MEDIANTE MACHINE LEARNING"**, el cual ha sido desarrollado para optar por el título de: Ingeniero en Biomedicina, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 29 de Febrero del año 2024

Atentamente,

Alex Fernando Tallana Castro

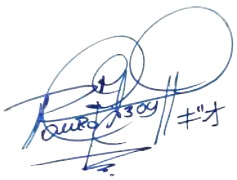
1751426444

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Luis Geovanny Romero Mejía con documento de identificación N° 1714731203, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **"DESARROLLO DE UN MODELO DE CLASIFICACIÓN AUTOMÁTICO EN EL SOFTWARE R PARA LAS ETAPAS DE LA ENFERMEDAD RENAL CRÓNICA MEDIANTE MACHINE LEARNING"**, realizado por Alex Fernando Tallana Castro con documento de identificación N° 1751426444, obteniendo como resultado final el trabajo de titulación bajo la opción Trabajo experimental que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 29 de Febrero del año 2024

Atentamente,



Ing. Luis Geovanny Romero Mejía

1714731203

Dedicatoria

A Dios por la guía, fortaleza y paciencia que me dio día a día en todo este tiempo. Por no dejar que me rinda nunca, por siempre darme una motivación y aprendizaje por medio las personas, amigos y seres queridos que he obtenido en el camino de mi vida.

A mi padre por que a pesar que el quería un camino diferente para mí, nunca dejo de apoyarme en todo el proceso. Él es un pilar fundamental para mi crecimiento como persona y futuro profesional. Me ha enseñado que podemos aprender de los errores del pasado para ser mejores personas en el presente.

A mi madre por que a pesar de la distancia siempre me brindo un apoyo incondicional. Sus palabras de motivación me ayudaron a tomar decisiones importantes en mi vida. Gracias a su apoyo he podido llegar hasta este punto, le agradezco infinitamente todo el esfuerzo que ella da por mi y por mis hermanos.

1. Resumen

En este proyecto se presenta el desarrollo de un modelo de clasificación automático para las etapas de la enfermedad renal crónica mediante algunas técnicas de machine learning, como las máquinas de soporte vectorial y las redes neuronales. Se utilizó la base de datos de UCI Machine Learning, con aproximadamente 400 instancias de pacientes con y sin ERC, para entrenar y probar el modelo.

Se utilizó el software R para el modelado, pero previo a esto se realizó el preprocesamiento de la base de datos con el fin de identificar los datos faltantes y atípicos en cada una de las variables. De igual forma, se realizó un análisis y un procesamiento de los datos en el cual se añadieron nuevas variables, como la tasa de filtrado glomerular estimada (TFGe), calculada con la ecuación CKD-EPI, para clasificar instancias según su valor. Luego, se seleccionaron características en base al nivel de correlación de las variables con la TFGe, con el objetivo de mejorar el rendimiento del modelo, buscando clasificar instancias en las cinco etapas de ERC.

De esta forma, se obtuvo como mejor modelo el uso de las ANN, con un total de 6 características, como la edad, urea en sangre, creatinina sérica, hemoglobina, volumen de células empaquetadas y la cantidad de células rojas. El modelo generó una precisión del 91 % y una sensibilidad del 93 % para clasificar instancias de prueba que no padecían ERC. En el caso de personas con etapas 1 y 2, el modelo tuvo un rendimiento por debajo del 70 %, por lo cual no era tan fiable en estas etapas. En el caso de la etapa 3, el modelo llego a una precisión del 76 % y una sensibilidad del 80 %. En cambio, en la etapa 4, se tuvo una precisión del 82 % y una sensibilidad del 95 %. Además, con una precisión del 100 % y una sensibilidad del 57 % para las personas que se encontraban en un fallo renal grave.

Palabras clave: Clasificador, Enfermedad Renal Crónica, Redes Neuronales, Máquina de Soporte Vectorial, Multiclase.

Abstract

This project presents the development of an automatic classification model for the stages of chronic kidney disease using some machine learning techniques, such as support vector machines and neural networks. The UCI Machine Learning database, with approximately 400 instances of patients with and without CKD, was used to train and test the model.

R software was used for modeling, but prior previously, the database was preprocessed in order to identify missing data and outliers in each of the variables. Similarly, data analysis and processing was performed in which new variables were added, such as the estimated glomerular filtration rate (eGFR), calculated with the CKD-EPI equation, to classify instances according to their value. Then, features were selected based on the level of correlation of the variables with eGFR, with the aim of improving the performance of the model, seeking to classify instances in the five stages of CKD.

Thus, the best model obtained was the use of ANNs, with a total of 6 features, such as age, blood urea, serum creatinine, hemoglobin, packed cell volume and the number of red blood cells. The model generated an accuracy of 91 % and a sensitivity of 93 % for classifying test instances that did not have CKD. For individuals with stage 1 and 2, the model performed below 70 %, making it not as reliable for these stages. In the case of stage 3, the model reached an accuracy of 76 % and a sensitivity of 80 %. On the other hand, in stage 4, it had an accuracy of 82 % and a sensitivity of 95 %. In addition, with an accuracy of 100 % and a sensitivity of 57 % for people in severe renal failure.

Keywords: Chronic Kidney Disease, Classifier, Vector Support Machine, Multiclass, Neural Networks.

Índice

Lista de Tablas

Lista de Figuras

2. INTRODUCCIÓN

2.1. Antecedentes

En 1960 ocurrió un acontecimiento importante en la historia de la enfermedad renal crónica (ERC). Gracias al Dr. Belding Kh. Scrivener se tuvo la posibilidad de salvar la vida de los pacientes mejorando la tecnología de hemodiálisis. Desde entonces, los esfuerzos se han centrado principalmente en el tratamiento a largo plazo y la sustitución de la función renal mediante diálisis y trasplante renal. En cambio, en la década de 1990, se evidenció una alta mortalidad en pacientes sometidos a diálisis, debido a comorbilidades y complicaciones relacionadas con la enfermedad renal, destacando en particular la falta de detección precoz de la enfermedad en sus etapas iniciales.(Martín de Francisco y cols., 2009).

Por lo tanto, surgió la necesidad de clasificar la ERC desde sus etapas iniciales hasta su etapa terminal, con el objetivo de lograr una detección precoz que permitiera reducir las complicaciones asociadas. Y sobre todo para poder identificar a las personas que se encontraban en una etapa terminal de la enfermedad, de forma que se las pueda dirigir de manera rápida hacia un nefrólogo para prepararlas para un tratamiento renal sustitutivo. Y no fue hasta el año 2002 que la National Kidney Foundation (NFK) impulsó la clasificación de la ERC basada en estadios de severidad de acuerdo al nivel de la tasa de filtrado glomerular (TFG). Según esta clasificación, se reconocen cinco etapas, que van desde la inicial, caracterizada por un daño renal leve, hasta la etapa terminal, que implica un fallo renal completo. (NKF, 2002).

En la práctica clínica para calcular la TFG, se emplean diversas fórmulas, entre las cuales las más comunes son la "Modificación de la Dieta en la Enfermedad Renal de 4 Variables"(MDRD4) y la Çolaboración entre Enfermedad Renal Crónica y Epidemiología"(CKD-EPI). Según algunas investigaciones recientes, se sugiere que la fórmula CKD-EPI supera a la MDRD4, ya que esta última se basa únicamente en datos de pacientes con función renal disminuida, mientras que CKD-EPI se desarrolló con un grupo más diverso, incluyendo individuos con función renal normal y disminuida. Es por eso que la fómula de CKD-EPI tiene una mayor capacidad de correlación con la tasa de filtrado glomerular en individuos sanos (Burballa y cols., 2018).

Actualmente, el Machine Learning (ML) o también denominado aprendizaje automático, es una de las ramas de la Inteligencia Artificial (IA) que se presenta como una alternativa crucial para el diagnóstico temprano y preciso de diversas enfermedades. El uso de algoritmos

de aprendizaje automático en el diagnóstico de la ERC ha ganado importancia, ya que permite realizar pronósticos certeros y tempranos, facilitando la estratificación del riesgo de los pacientes. La eficacia de estas tecnologías respaldadas por algoritmos computacionales avanzados se ha demostrado en diversas áreas médicas, proporcionando soporte en la toma de decisiones y en el diagnóstico de enfermedades (Vidal y Vidal, 2022; Xie y cols., 2019).

En los últimos años se han realizado investigaciones de la predicción de la ERC, este tipo de investigaciones únicamente se centraron en una predicción de tipo binomial, es decir llegar a determinar si un paciente tiene el riesgo de padecer ERC o de estar sano. Por ejemplo en un estudio, se utilizó redes neuronales (ANN) como clasificador para predecir la enfermedad renal crónica en la población colombiana, el modelo de la ANN alcanzó una precisión del 95 %. Además, se evaluaron otras métricas como sensibilidad, especificidad, precisión, exhaustividad, valor-f y el área bajo la curva (AUC) para determinar la calidad del modelo (Morales y Ricardo, 2019).

En otra investigación, se usaron estrategias de selección de características basadas en enjambre de partículas con datos del repositorio UCI Machine Learning. Se evaluaron seis modelos de ML, y los que incluyeron selección de características tuvieron un mejor desempeño. El clasificador basado en Máquinas de Soporte Vectorial (SVM) destacó como el mejor método, con una sensibilidad del 90.91 %, especificidad del 87.50 %, valor-F del 93.02 %, precisión del 90.00 %, y una tasa de error del 10.00 % (Belina y K, 2018).

Para el caso de investigaciones que tenian como objetivo la predicción de las etapas de la ERC, se encontraron escasos estudios con metodologías sólidas. Un ejemplo de esto es el estudio que utilizaron la base de datos de UCI Machine Learning para crear varios modelos, entre los cuales el más sobresaliente resultó ser el de Redes Neuronales Probabilísticas (PNN). El modelo PNN tuvo el mayor porcentaje de precisión general con un valor del 96.7 %. La gran ventaja que tiene este estudio es que brinda las métricas como la sensibilidad, especificidad, precisión, recall, valor-f, entre otras, de cada una de las etapas de la ERC.

Sin embargo esta investigación, tiene dos puntos desfavorables, uno es que, para el cálculo de la TFG hacen uso de la ecuación MDRD4 y la otra es que en la división de entrenamiento del modelo se incluyo a todas las personas sanas dentro de las personas con ERC en una etapa inicial, específicamente en la etapa 1. Es decir, no consideraron una clase de personas sanas para su modelo, lo cual puede generar un sesgo tal que las capacidades del mismo no lograrán dife-

renciar adecuadamente las características entre la población sana y la afectada por ERC en sus primeras etapas, afectando así la validez y utilidad de sus clasificaciones (Rady y Anwar, 2019).

Esta área de aplicación abre un campo de investigación interesante sobre el uso de los modelos computacionales de IA para la toma de decisiones, lo que evidencia que aún existe mucho trabajo por desarrollar para determinar la efectividad y el buen desempeño de estas estrategias.

2.2. Problemática

La ERC es un desafío de salud pública a escala global, ya que a lo largo del tiempo ha habido un incremento en la prevalencia de esta enfermedad. En la actualidad, aproximadamente 850 millones de individuos se ven afectados por esta enfermedad a nivel mundial (García-Maset y cols., 2022). Estimándose, que será la quinta causa de muerte prematura más común para el año 2040 (Sánchez, Guacho, y Guerrero, 2021). Además esta enfermedad se le considera como una enfermedad silenciosa, dado que no presenta síntomas hasta que se encuentre en una etapa avanzada en la que puede llevar a la muerte de una persona que la padezca, por tal razón la Organización Panamericana de la Salud (OPS) declára que es de suma importancia su diagnóstico precoz (OPS, 2015).

En el contexto de Ecuador, según el Ministerio de Salud Pública (MSP) la ERC figura como la cuarta causa de mortalidad general. Además, el Instituto Nacional de Estadística y Censos (INEC) ha registrado las enfermedades del sistema urinario como una de las 10 principales causas de muerte en el país, ocupando el octavo lugar en dicha lista (INEC, 2020).

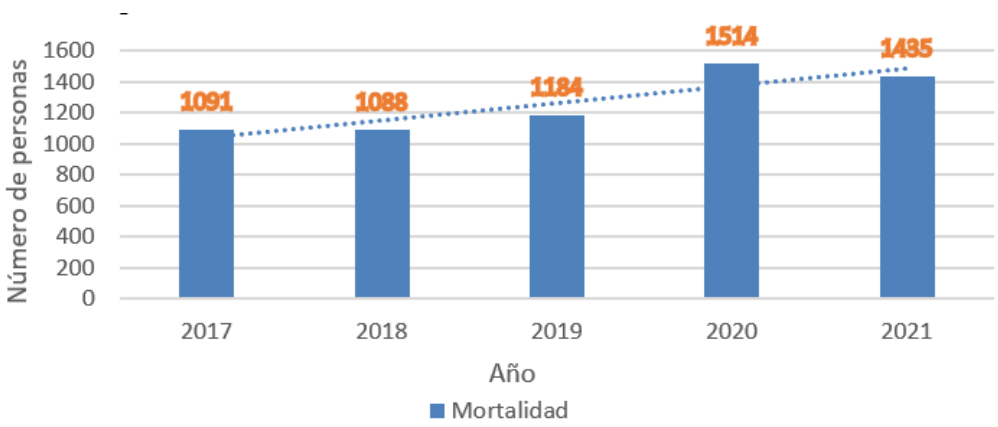
La mortalidad relacionada con la ERC ha experimentado un preocupante aumento en los últimos años. Según los datos representados en la figura 1, se puede apreciar una tendencia lineal en constante aumento desde el año 2017 hasta el 2021. Este aumento sostenido en la mortalidad por ERC plantea desafíos significativos para la salud pública y pone de manifiesto la importancia de abordar esta afección de manera más efectiva.

Así mismo, no solo la mortalidad ha aumentado, sino que también la prevalencia de la enfermedad ha ido en constante aumento. En 2020, se registraron 6,099 casos de ERC, y para el año 2022, este número se incrementó considerablemente a 8,904 casos, como se ilustra en la figura 2. Este incremento también sigue una tendencia lineal creciente en ese periodo.

Este aumento en la prevalencia plantea la necesidad de una atención más sólida y enfoques preventivos para hacer frente a la creciente carga de la ERC en la población.

Figura 1

Mortalidad por ERC en Ecuador.



Nota: Estadísticas de la Mortalidad por ERC desde el año 2017 hasta el 2021. Fuente: (INEC, 2020).

Figura 2

Egresos Hospitalarios por ERC en Ecuador.

Nota: Estadísticas de la Prevalencia de la ERC desde el año 2017 hasta el 2022. Fuente: (INEC, 2022).

Hasta el momento, existen pocos trabajos que hayan abordado la predicción de las etapas de la ERC. En todos estos estudios, se ha utilizado la ecuación MDRD4 para calcular la Tasa de Filtrado Glomerular Estimada (TFGe). Sin embargo, en la actualidad, es ampliamente reconocido que esta ecuación tiende a subestimar los valores en individuos con una TFGe elevada y puede categorizarlos como si tuvierán una disminución de la función renal (TFG <60 ml/min) a individuos sin patología renal.

Además, es importante señalar que en nuestra nación, actualmente contamos únicamente con pautas para el diagnóstico, control y tratamiento, pero carecemos de enfoques específicos orientados hacia modelos de pronóstico o clasificación, dado que todos de los estudios mencionados se realizarón fuera del país.

2.3. Justificación

Teniendo en cuenta todas las dificultades que la ERC representa tanto para las personas ecuatorianas como para la población a nivel mundial, La Sociedad Ecuatoriana de Nefrología (SEN) mencionan que la prevención de la ERC por medio de las intervenciones habituales son menos eficaces y que es necesario tanto investigar y aplicar nuevas estrategias como

para poder realizar la prevención precoz de la enfermedad. Algo similar sostiene la OPS, ya que menciona la detección temprana de la enfermedad como una estrategia para iniciar un tratamiento inmediato y evitar el avance progresivo de la enfermedad, además de fortalecer el conocimiento de la situación en cada país (OPS, 2015).

Por tal razón la presente propuesta de titulación tiene el objetivo de desarrollar un modelo de clasificación automático en el software R para las etapas de la enfermedad renal crónica utilizando técnicas de Machine Learning (ML). Para llevar a cabo este proyecto, se utilizará la base de datos de acceso público de la Universidad de California Irvine (UCI), la cual ha sido diseñada específicamente para facilitar estudios y proyectos de Machine Learning de forma que se desarrollen modelos de clasificación y predicción en el contexto de esta enfermedad (UCIrvine, 2015).

En este estudio, a diferencia de muchos otros, se realizará el uso de la ecuación CKD-EPI dentro de la base de datos de UC Irvine Machine Learning. Esta elección se justifica por el hecho de que esta ecuación no ha sido previamente empleada en esta base de datos. Además, se opta por esta ecuación debido a su mayor actualidad de desarrollo en comparación con la MDRD4. Al ser más nueva que la MDRD4 tiene una mejor exactitud y precisión de la estimación de la TFGe, sobre todo para valores superiores a 60 ml/min, de esta forma al trabajar con esta ecuación se tendra un modelo más eficaz y sobre todo un modelo que será entrenado con datos más apegados a la realidad de los pacientes de la base de datos.

2.4. Objetivos

2.4.1. Objetivo General

Desarrollar un modelo de clasificación automático utilizando técnicas de Machine Learning en el software R para las etapas de la enfermedad renal crónica.

2.4.2. Objetivos Específicos

- Procesar los datos obtenidos del repositorio de UC Irvine Machine Learning con el fin de mejorar la convergencia de los modelos.
- Entrenar modelos de Redes Neuronales y Máquinas de Soporte Vectorial aplicando métodos de validación para la optimización del rendimiento del modelo.

- Evaluar el desempeño de los modelos de clasificación mediante un conjunto de datos de prueba de la base de datos para obtener el mejor modelo.

3. Hipótesis

3.1. Hipótesis General

El desarrollo de un modelo de clasificación automático de las etapas de la enfermedad renal crónica ayudará a clasificar correctamente las etapas de la ERC.

3.2. Hipótesis Específicas

- El procesamiento de los datos mejorará la convergencia de los modelos.
- Aplicar métodos de validación a los modelos de aprendizaje automático optimizará el rendimiento de los modelos.
- Se encontrará el mejor modelo, evaluando el desempeño de los mismos mediante un conjunto de datos de prueba de la base de datos pública.

4. Fundamento Teórico

4.1. Enfermedad Renal Crónica (ERC)

La ERC se define como una condición patológica, que ocurre a lo largo de meses o años, en la que su característica distintiva es la reducción progresiva de la función renal, lo cual implica la capacidad de los riñones para eliminar los productos de desecho metabolico del cuerpo (Malkina, 2023).

4.1.1. Ecuaciones para el cálculo de la Tasa de filtrado glomerular estimada (TFGe)

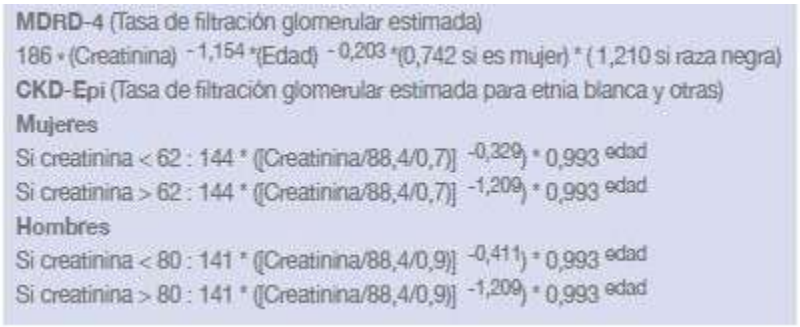
La cuantificación del filtrado glomerular se puede realizar utilizando marcadores externos que sirven como indicadores de la función renal. Entre estos marcadores se encuentran los contrastes radiológicos, como por ejemplo, el iotalamato, diatrizoato e iohexol, los cuales han sido respaldados por varios estudios que confirman su utilidad como marcadores de la TFG (Evia, 2008). Sin embargo, estos marcadores presentan desventajas, ya que los procedimientos en los que se los usa toman demasiado tiempo y son de alto costo. Además, se pueden tener

complicaciones técnicas que impiden su aplicación en el día a día de la de la práctica clínica (Vilche, Alejandro, y Correa, 2022). Es por esto que, debido a esas dificultades, se han creado algunas ecuaciones para calcular la TFGe, tal como se observa en la figura 3.

- **MDRD4:** Esta ecuación fue desarrollada en 1999 con datos de 1628 individuos con ERC. La validación de la ecuación fue solo en pacientes enfermos, por lo cual esta ecuación subestima los valores en personas con una TFG elevada. Es por esto que esta ecuación puede llegar a diagnosticar una disminución de la función renal en individuos sin afecciones renales (Vilche y cols., 2022).
- **CKD-EPI:** Esta ecuación se desarrolló en el 2009 y en esta ocasión sí se tuvo un grupo de datos mas amplio, dado que se trabajó con 8254 individuos con y sin ERC. Esta ecuación permite conseguir resultados más precisos, especialmente para valores de TFG superiores a 60 ml/min/1,73 m2. Es por esto que algunos expertos sugieren su integración a la práctica clínica común (Verdejo y cols., 2014).

Figura 3

Ecuaciones para cálculo de la TFGe



Nota: Ecuaciones usadas para el cálculo de la TFGe. Fuente: (Verdejo y cols., 2014).

4.1.2. Etapas de la ERC

La severidad de la ERC se clasifica en cinco etapas según la TFGe, variable que proviene de fórmulas basadas en la creatinina sérica, edad y género (AKF's, 2023). En las etapas iniciales, los riñones conservan su capacidad para eliminar los residuos sanguíneos. A diferencia de las etapas más avanzadas, donde esa misma función se ve comprometida, e inclusive se tiene un

riesgo de la pérdida total de la función renal. En la tabla 1 se puede observar cada una de las etapas y una breve descripción de las mismas.

Tabla 1

Etapas de la Enfermedad Renal Crónica

Categoría ERC	TFGe (ml/min)	Descripción
G1	>=90	Normal o elevado
G2	60 – 89	Ligeramente disminuido
G3a	45 – 59	Ligera o moderadamente disminuido
G3b	30 – 44	Moderara o gravemente disminuido
G4	15 - 29	Gravemente disminuido
G5	<15	Fallo renal

Nota: Se puede observar que existen 5 categorías y cada uno depende del filtrado glomerular. Fuente: (Sellarés y Rodríguez, 2023).

4.1.3. Tratamientos de la ERC

El tratamiento de la ERC consta de varios puntos. Por ejemplo, abordar las causas particulares de la enfermedad renal, reconocer y solucionar las causas reversibles de daño renal, tratar a los elementos que contribuyen al avance de la enfermedad. En el caso de que la capacidad de eliminar productos de desecho ya se vea comprometida, como ocurre desde la etapa tres a la cinco, se buscará eliminar las toxinas que el riñon no puede filtrar (Rodrigo Orozco, 2010). Para el último caso las opciones incluyen diálisis (hemodiálisis o peritoneal), tratamiento conservador para aliviar síntomas y trasplante de riñón (NIDDKD, 2018).

4.1.4. Factores asociados a la ERC

La detección temprana de esta enfermedad es dificil, dado que no produce síntomas en sus etapas iniciales. Es por esto que es de importancia saber reconocer cuales son algunos factores de riesgo asocidados a la ERC (Guzmán, Fernández, Mora, y Vintimilla, 2014).

Por ejemplo, en la tabla 2 se puede observar que los mismos factores de riesgo asociados con la enfermedad cardiovascular (ECV) se tienen en la ERC, tanto en prevalencia, morbilidad y mortalidad. Y esto se debe a que la acumulación de hipertensión arterial (HA), diabetes y dislipidemia pueden causar un daño progresivo en los vasos sanguíenos y capilares de los riñones, lo cual afectara directamente en la capacidad de filtración y eliminación de desechos, reducción de flujo, etc.

Tabla 2

Evaluación de la ERC como factor de riesgo de ECV

Factor de riesgo de ECV	Prevalencia de ERC	Factor de riesgo de morbilidad - mortalidad
Hipertensión	Aumento	Aumento
Diabetes	Aumento	Aumento
Dislipidemia	Aumento	Aumento

Nota: ECV; Enfermedad cardiovascular. Fuente: (A. Levey y cols., 2007).

4.2. Aprendizaje automático

El aprendizaje automático, floreció con éxito en los años de 1950 junto con la ciencia de datos. Esta área de estudio forma parte de la inteligencia artificial y dentro de la misma se emplea algoritmos sobre conjuntos de datos con el objetivo de realizar predicciones en base a la cantidad y calidad de la información que se le proporcione (KYOCERA, 2023). La importancia del ML es que posibilita la automaticazión de tareas complicadas, en las que previamente se necesitaba la acción humana, lo cual optimiza tiempo y recursos, a la vez que se pueden tomar decisiones rápidas y respaldadas en datos (Lamorte, 2023).

4.2.1. Aprendizaje supervisado

En el aprendizaje supervisado, los algoritmos se centran en el uso de datos que tienen etiquetas específicas. El objetivo de este aprendizaje es encontrar una función que, basándose en las variables de entrada o también conocida como predictoras, pueda atribuir una variable de respuesta o salida (Simeone, 2018). La clave aquí es que previamente nosotros sabemos cual es valor que va a tomar la variable objetivo, y es por eso que se lo conoce como supervisado. Déspues de la predicción, se podra verificar bajo supervisión si el dato predicho se acerca o es

el mismo que el valor real.

Este aprendizaje se lo puede aplicar según las necesidades del estudio, dado que se tiene a los algoritmos de clasificación cuando tratamos con variables categóricas y por el otro lado estan los de regresión cuando de variables cuantitativas se trata (Santos, 2021).

4.2.2. Aprendizaje no supervisado

Dentro del aprendizaje no supervisado, ahora los datos ya no estarán procesados ni etiquetados. En este aprendizaje, el computador aprenderá a reconocer patrones y comprender procesos difíciles sin la necesidad de la intervención directa de un humano. Es por esto que el aprendizaje no supervisado se enfoca principalmente en identificar conjuntos similares dentro de los datos, también conocido como clustering o segmentación (Nowak, 2023).

4.2.3. Aprendizaje automático en el diagnóstico de enfermedades

El uso del ML en la atención médica ha llevado al desarrollo de software, plataformas y sistemas automatizados, abordando de manera efectiva la verificación y mejora del estado de salud de las personas. Estos avances son esenciales ya que el análisis de datos clínicos utilizando ML no solo permite diagnósticos oportunos, sino que también facilita la intervención temprana en el tratamiento del paciente (Massaro y cols., 2020).

Identificar una enfermedad en sus primeras etapas significa mitigar el efecto adverso que puede tener en la población. Este proceso realmente mejora la calidad de vida de los pacientes, mediante la administración de tratamientos adecuados, adaptados según la etapa en la que se encuentre la enfermedad (Abdar, Zomorodi-Moghadam, Das, y Ting, 2017).

Por ejemplo en un estudio de revisión, llegaron a determinar que los modelos basados en algoritmos de aprendizaje automático en la atención de la Diabetes Mellitus tipo 2 (DM2) se han enfocado en los resultados de diagnóstico, detección de prediabetes, la categorización de complicaciones y tambien el análisis de factores de riesgo (Shahabeddin, R., Mehdi, Hajar, y Ali, 2019).

Asi mismo se ha visto en el Enfermedad del Parkinson (EP) una necesidad en la predicción y un diagnóstico temprano para un tratamiento eficaz de esta enfermedad. Por lo cual se han recurrido a modelos de ML donde existe una gran variabilidad de fuente de datos, que

van desde grabaciones de voz hasta la combinación de datos de metabolómica sanguínea, entre otras. En resumen, la aplicación de ML en este tipo de patologías ha demostrado tener beneficios en el tratamiento y la gestión de la EP, aun así se deben corregir cuestiones como la exactitud y la precisión (Gupta, Kumari, Senapati, Ambasta, y Kumar, 2023).

Otro ejemplo del uso del ML es en el caso del diagnóstico temprano y preciso de la enfermedad de Alzheimer (EA), para esta enfermedad entre los diferentes datos y modalidades de imagen que se usan en los modelos de ML, se encuentra las imágenes proveninetes de estudios como la resonancia magnética (MRI), secuencia de protenínas, señales EEG, datos del habla e historial médico. A pesar de todas las investigaciones a día de hoy se siguen realizando investigaciones para poder proporcionar un enfoque eficiente y preciso para el diagnóstico y la predicción de la EA (Mirzaei y Adeli, 2022) (Mirzaei y Adeli, 2023).

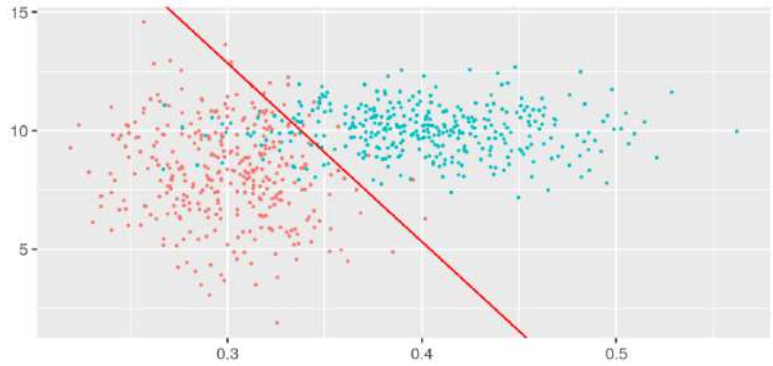
4.2.4. Problemas de clasificación

La clasificación se puede abordar con el aprendizaje automático y se encuentra dentro del aprendizaje supervisado. Dentro del mismo se disponen de bases de datos que contienen varias variables etiquetadas e instancias que son de tipo categóricas de dos o más clases y tambien contienen variables cuantitativas (Diez, 2023). Al hablar de problemas de clasificación, la variable a predecir o a estimar, será de tipo categórica, esta variable posiblemente tendrá alguna correlación entre las demas variables de la base de datos, esa relación entre variables es la que ayudará a que el modelo de clasificación sea bueno o malo. Entonces el objetivo en este tipo de problemas sera construir un clasificador capaz de asignar o estimar la clase de un ejemplo o instancia desconocido en base a una serie de variables predictoras.

Por ejemplo en la figura 4, se puede observar un problema de clasificación binomial, que se encarga de separar dos clases por medio de una línea recta. A este modelo se lo conoce como una regresión logística. El objetivo de este modelo es obtener una función logística, como se muestra en la figura 5, que permita clasificar individuos en grupos, donde el número de grupos está determinado por las categorías de la variable dependiente (KeepCoding, 2022).

Figura 4

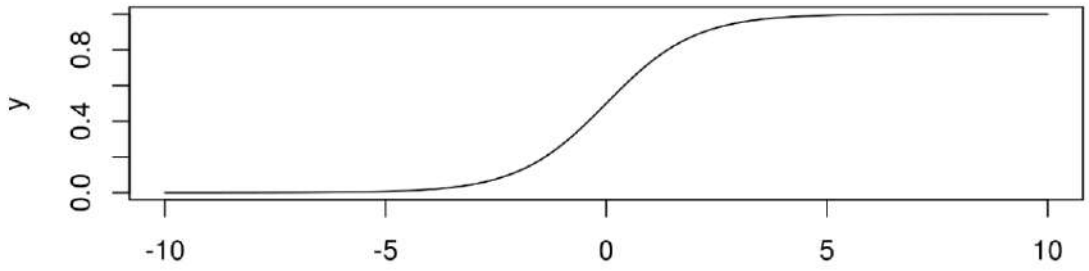
Regresion Logística



Nota: Trazado de una recta que separa dos clases de una variable dependiente. Fuente: (KeepCoding, 2022).

Figura 5

Función Logística



Nota: Se tiene una probabilidad de que el valor estimado de la variable dependiente este en un rango de [0,1]. Fuente: (KeepCoding, 2022).

4.2.5. Evaluación de modelos clasificadores

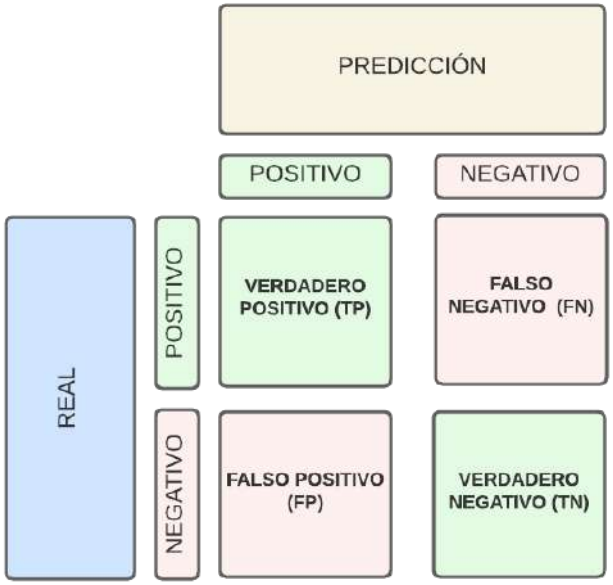
Una manera de presentar los resultados de predicción de un clasificador es usar una matriz de confusión. La dimensión de la matriz de confusión dependerá de la cantidad de clases que tenga la variable a predecir, es decir si es de k clases, tendrá una dimensión de k x k. En la parte superior de la matriz se tendran a las etiquetas de clase estimada y en el lado izquierdo

las clases reales (Aucejo, 2022).

Dentro de la matriz de confusión de la figura 6, tenemos a los verdaderos positivos y los verdaderos negativos. Estos representan el numero de veces que el modelo predice adecuadamente la clase positiva y negativa respectivamente. En el caso de los falsos positivos, se refiere las veces que el modelo se equivocó en la clasificación de una clase que era negativa pero la toma como positiva. Así mismo, se tiene a los falsos negativos, que no son más que clases que en realidad eran positivas pero fueron estimadas como si fuesen negativas.

Figura 6

Matriz de Confusión



Nota: Representación de una matriz de confusión. Elaborado por: (El autor, 2024)

A partir de la matriz de confusión nostros podemos sacar algunas métricas de importancia que nos diran segun su valor que tan bueno es mi modelo de clasificación. Estas son (Arce, 2019) :

- **Sensibilidad/Recall/ Recuperación/ True Positive Rate (TPR):** Mide la proporción de Verdaderos Positivos que se identifican correctamente como tales.

$$Sensibilidad = \frac{TP}{TP + FN}$$

- **Especificidad/ True Negative Rate (TNR):** Mide la proporción de negativos que se identifican correctamente como tales.

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

- **Precisión:** Es la proporción de ejemplos que realmente pertenecen a la clase de entre los que fueron clasificados como pertenecientes a la clase.

$$Precisión = \frac{TP}{TP + FP}$$

- **Accuracy/ Exactitud:** Es el porcentaje de aciertos del modelo con respecto a todos los datos.

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN}$$

- **F-Measure:** Esta métrica transmite el equilibrio entre la precisión y la recuperación. Un puntaje alto de F1 significa que tiene bajos falsos positivos y negativos, por lo que está identificando correctamente algun tipo de clase y casi no comete errores.

$$PuntajeF = \frac{2 * Precisión * Sensibilidad}{Precisión + Sensibilidad}$$

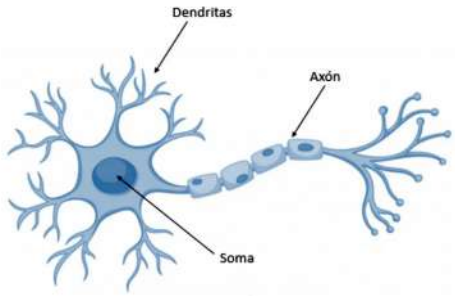
4.3. Algoritmos de apredizaje automático

4.3.1. Redes neuronales

Una red neuronal se inspira en el procesamiento de información en el cerebro humano, donde las dendritas recogen impulsos nerviosos, el soma los procesa y el axón los transmite a otras neuronas, como se ilustra en la figura 7.

Figura 7

Neurona

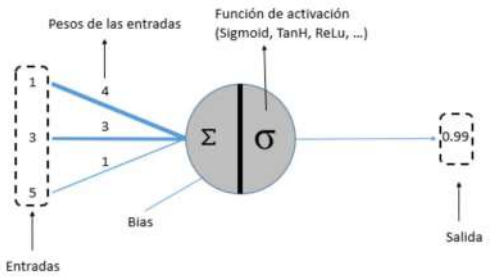


Nota: Representación de una neurona. Fuente: (Olivera, 2019).

Por otra parte una neurona artificial se representa en la figura 8. En este caso el impulso nervioso esta determinado por la sumatoria de las entradas y mutliplicadas por los pesos asociados. Este valor se procesará dentro del soma gracias a la función de activación que tendrá como salida un valor que se enviará a la salida de la neurona.

Figura 8

Red neuronal



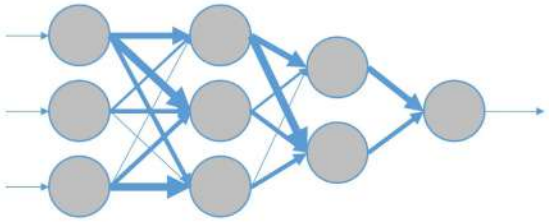
Nota: Representación de una red neuronal. Fuente: (Olivera, 2019).

Tal como en el cerebro, las neuronas artificiales se interconectan y organizan en capas. La primera capa, llamada capa de entrada, recibe los datos originales introducidos en la red. La última capa, conocida como capa de salida, produce el resultado final de la red. Las capas in-

termedias entre la capa de entrada y la de salida se denominan capas ocultas porque los valores de entrada y salida en estas capas no son directamente observables o conocidos (Olivera, 2019).

Figura 9

Red neuronal con cuatro capas



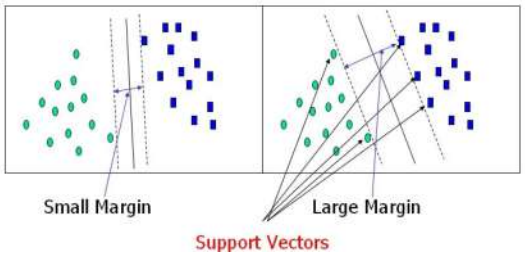
Nota: Representación de una red neuronal con cuatro capas. Fuente: (Olivera, 2019).

4.3.2. Máquinas de soporte vectorial (MSV)

Este algoritmo se utiliza para problemas de clasificación o regresión. En el contexto de clasificación, actúa como un clasificador discriminativo cuando se trata de dos o más clases de datos etiquetados. Utiliza un hiperplano de separación que permitirá diferenciar las clases, como se muestra en la figura 11 (Rodriguez, 2020). Finalmente quien determina la posición del hiperplano serán los vectores de soporte, los cuales no son más que los datos con los que se entrená al modelo. De igual forma se debe considerar el costo, el cual determinará la penalización que se le dara al margen del hiperplano, a medida que sea cercano a 0, el margen será más ancho y a medida que se aleje de 0 menor será el margen.

Figura 10

Máquina de soporte vectorial

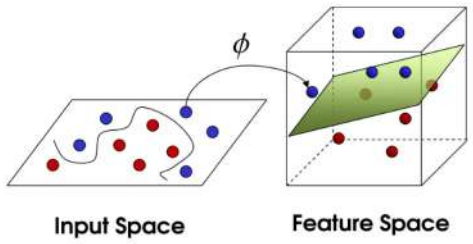


Nota: Representación de los vectores de soporte. Fuente: (Rodriguez, 2020).

En los modelos de SVM se debe tomar en cuenta el tipo de kernel a emplear. Estos kernels operan al mapear los datos a un espacio de mayor dimensión, lo que facilita la separación de clases y simplifica las fronteras de decisión que, de otra manera, serían complejas y no lineales en el espacio de características original. Este proceso se llama truco de kernel y evita la necesidad de transformar los datos de manera explícita (MatLab, S.f).

Figura 11

Truco del kernel



Nota: Representación del truco del kernel. Fuente: (Rodriguez, 2020).

5. Análisis y Procesamiento de la base de datos

5.1. Base de datos

La base de datos del repositorio de UCI Machine Learning fue recopilada en la India en un periodo de dos meses. Tiene un total de 25 características y cuenta con la información de 400 instancias o individuos (UCIrvine, 2015). En la Tabla 3 se puede observar cada una de las variables, sus descripciones y sus unidades.

Tabla 3

Variables de la Base de Datos de UCI Machine Learning

Indice	Variable	Tipo	Unidades/Rango
1	Edad	Numérica	Edad en años
2	Presión Arterial	Numérica	mm/Hg
3	Gravedad Específica	Nominal	(1.005,1.010, 1.015, 1.020, 1.025)
4	Albumina	Nominal	(0,1,2,3,4,5)
5	Azucar	Nominal	(0,1,2,3,4,5)
6	Células rojas	Nominal	(normal,anormal)
7	Células de pus	Nominal	(normal,anormal)
8	Aglomeraciones de células de pus	Nominal	(normal,anormal)
9	Bacteria	Nominal	(Presente, No Presente)
10	Glucosa en sangre aleatorio	Numérica	mgs/dl
11	Urea en sangre	Numérica	mgs/dl
12	Creatinina sérica	Numérica	mgs/dl
13	Sodio	Numérica	mEq/L
14	Potasio	Numérica	mEq/L
15	Hemoglobina	Numérica	gms
16	Volumen de células empaquetadas	Numérica	mL
17	Células blancas	Numérica	cells/cumm
18	Células rojas	Numérica	millions/cmm
19	Hipertensión	Nominal	(Sí,no)
20	Diabetes Mellitus	Nominal	(Sí,No)
21	Arteriopatóa coronaria	Nominal	(Sí,No)
22	Apetito	Nominal	(Bueno,Pobre)
23	Edema pedal	Nominal	(Sí,No)
24	Anemia	Nominal	(Sí,No)
25	Clase	Nominal	(Con ERC, Sin ERC)

Nota: Se puede observar que existen 5 categorías o estados y cada uno depende del filtrado glomerular. Fuente: (UCIrvine, 2015).

5.2. Preprocesamiento de los datos

5.2.1. Datos faltantes

Al igual que muchas bases de datos de repositorios de acceso abierto, esta base de datos posee datos faltantes o nulos. Antes de realizar cualquier modelo de ML es importante primero que tratar esos datos y de igual forma llegar a determinar que el porcentaje de datos faltantes no supere el 20 %, para no poner en riesgo la confianbilidad de las variables (Medina y Galván, 2007).

En la tabla 4, se observa la cantidad de valores faltantes por cada variable. El total de datos faltantes en toda la base de datos asciende a 1015, lo que representa un 10.15 % de pérdida de información con respecto al total de datos, que es de 10,000.

Tabla 4

Datos faltantes por variable de la base de datos

Variable	Datos faltantes	Variable	Datos faltantes
Edad	9	Sodio	87
Presión Arterial	12	Potasio	88
Gravedad Específica	47	Hemoglobina	52
Albumina	46	Vol. de células empaquetadas	71
Azucar	49	Células blancas	106
Células rojas	152	Células rojas	131
Células de pus	65	Diabetes Mellitus	3
Aglomeraciones de células de pus	4	Hipertensión	2
Bacteria	4	Arteriopatóa coronaria	2
Glucosa en sangre aleatorio	44	Apetito	2
Urea en sangre	19	Edema pedal	2
Creatinina sérica	17	Anemia	1

Nota: Se tiene un total 1015 datos faltantes entre todas las variables. Elaborado por: (El autor, 2024).

5.2.2. Datos atípicos

Algo importante, además de identificar la cantidad de datos faltantes, es saber reconocer a los outliers o datos atípicos dentro de la base de datos. Dado que si primero se realiza la imputación de los datos, junto con datos faltantes, también se imputaría un porcentaje de datos atípicos.

Para encontrar los datos atípicos se usó el diagrama de caja y bigotes, el cual es bastante útil para identificar de forma visual en qué variables se tiene la mayor cantidad de datos atípicos. De igual forma, se determinó la media, mediana y desviación estándar con el fin de ver qué tanto afectan los datos atípicos a algunas métricas de tendencia central y de dispersión.

Tabla 5

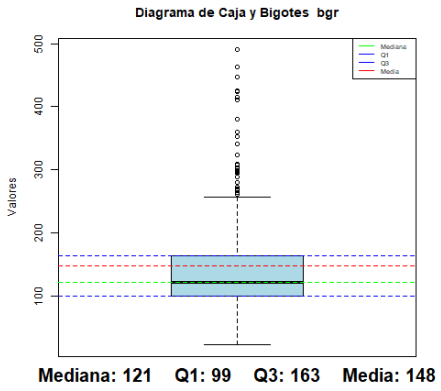
Medidas de tendencia central y dispersión

Variable	Media	Mediana	Desviación Estándar
Edad	51.5	55	17.17
Glucosa en sangre aleatorio	148	121	79.28
Presión Arterial	76.5	80	13.68
Urea en sangre	57.4	42	50.5
Hemoglobina	12.5	12.6	2.91
Volumen de células empaquetadas	38.9	40	8.99
Potasio	4.6	4.4	3.19
Células rojas	4.7	4.8	1.03
Creatinina sérica	3.1	1.3	5.74
Sodio	137.5	138	10.41
Células blancas	8406	8000	2944.47

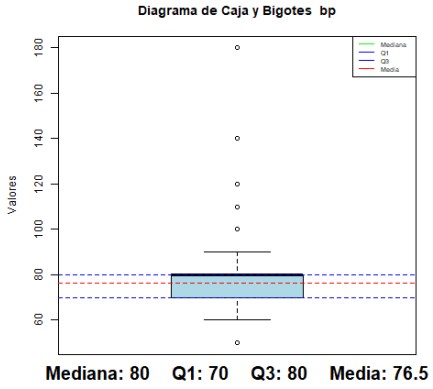
Nota: Medidas de tendencia central y dispersión a las varibales cuantitativas. Elaborado por: (El autor, 2024).

Figura 12

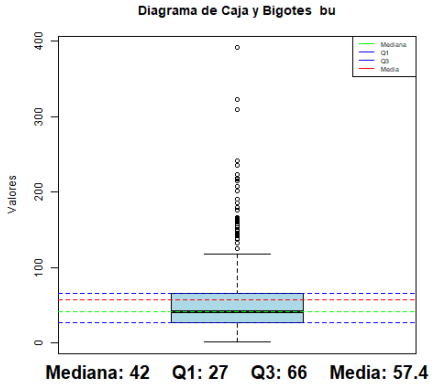
Diagrama de caja y bigotes de las varaiables cuantitativas originales



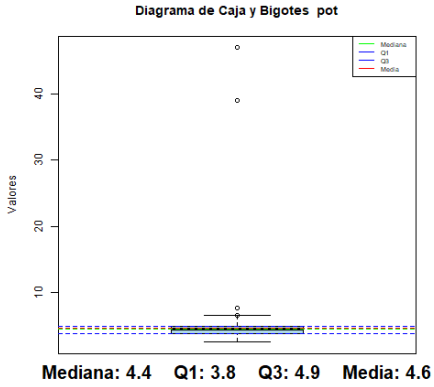
(a) *Glucosa en sangre aleatorio*



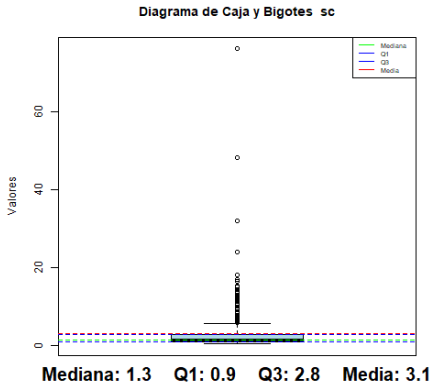
(b) *Presión Arterial*



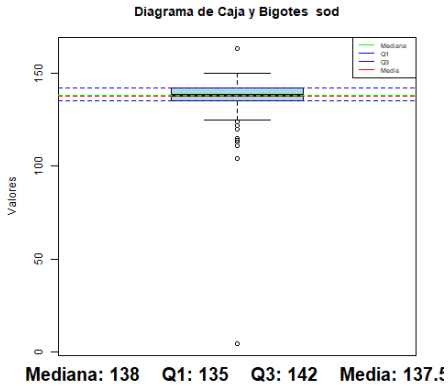
(c) *Urea en sangre*



(d) *Potasio*



(e) *Creatinina sérica*



(f) *Sodio*

Nota: Elaborado por: (El autor, 2024).

A pesar de que todas las variables cuantitativas tienen outliers, en la figura 12 se representan los diagramas de caja y bigotes de las variables más afectadas por los outliers. El proceso que se usó para determinar los outliers de las variables, fue por medio del rango intercuartílico, todos los datos que estaban fuera de este rango fueron considerados como menos probables y, por lo tanto, se clasificaron como outliers. De igual forma según nuestros gráficos y los cálculos de las medidas de tendencia central y dispersión de la tabla 5, las variables con más valores atípicos tendían hacia una mayor desviación estándar y a su vez tenían una mayor diferencia entre su media y mediana.

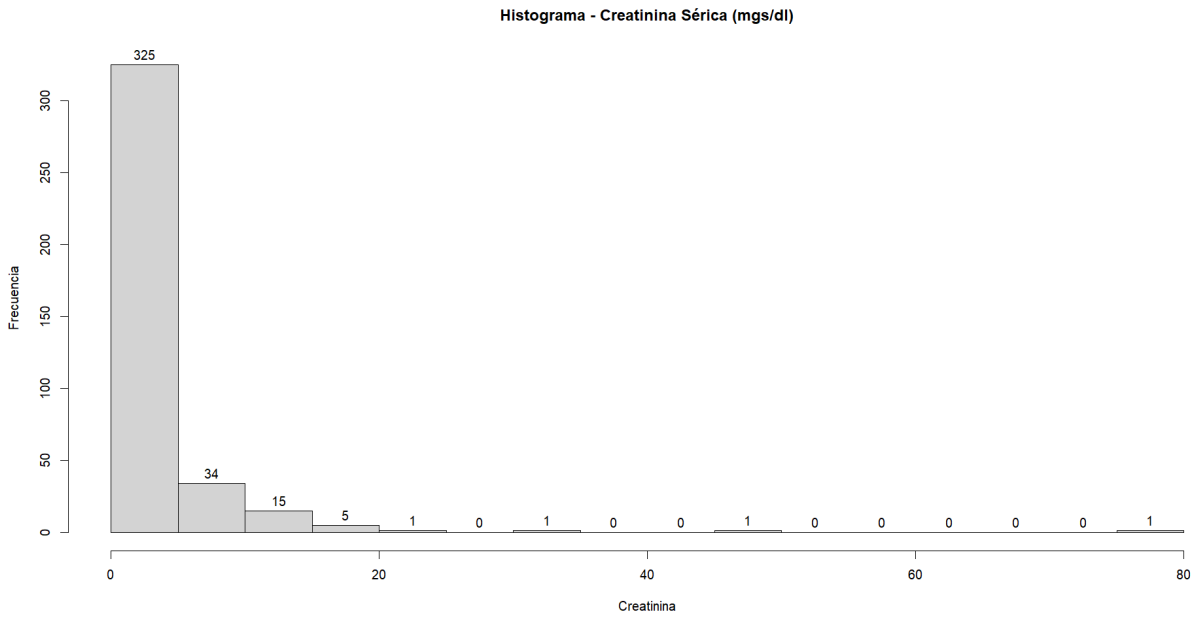
La media al ser más sensible a los outliers, se decidió tratar a esos valores atípicos por medio de la mediana. La mediana al representar de mejor forma la tendencia central que toman los datos afectados por outliers, nos dará una mejor representación de los datos hacia una realidad presente en la práctica clínica y la ERC.

Por ejemplo, analizando cómo se distribuyen los datos de la creatinina sérica, la cual será fundamental más adelante para el cálculo de la TFGe. Originalmente la creatinina sérica iba desde valores de 0.4 hasta 76 mgs/dl, tal como se observa en la figura 13 (a). Tener un valor de 76 mgs/dl es algo muy inusual. De hecho según un estudio publicado en el 2021, la creatinina sérica más alta registrada a nivel mundial fue de 73.8 mgs/dl y se trataba de un paciente con antecedentes de transplante de riñon que presento una falla renal grave (Persaud y cols., 2021).

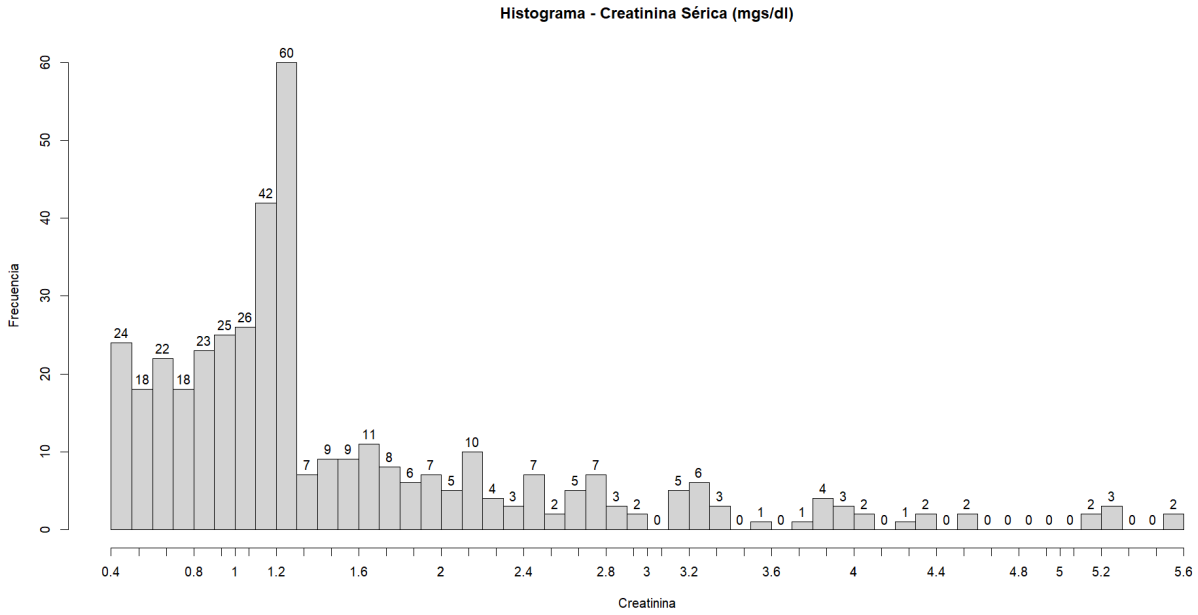
Ahora, analizando la figura 13 (b), después de tratar los outliers dentro de esta variable vemos que su rango cambio de 0.4 a 5.6 mgs/dl, el cual es un rango más aceptable y que engloba a pacientes sanos y pacientes con ERC desde la primera hasta la quinta etapa.

Figura 13

Histograma Creatinina Sérica



(a) *Histograma de la variable sin tratar outliers*



(b) *Histograma de la variable tratando outliers*

Nota: En el gráfico (b) se tiene un mejor rango y más acercado a la práctica clínica. Elaborado por: (El autor, 2024).

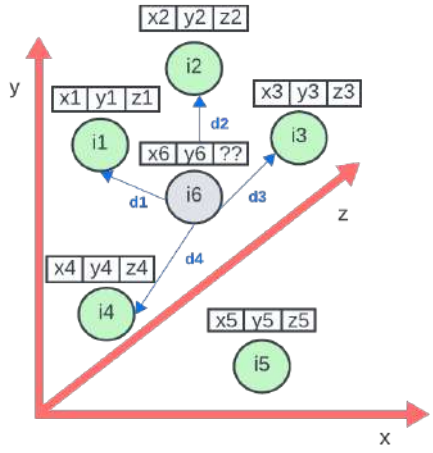
5.2.3. Imputación de datos

Una vez que se haya tratado los datos de los outliers ahora sí se puede realizar el proceso de imputación. Para realizar la imputación de los datos existen varias formas, ya que podemos imputar datos faltantes por la media, mediana o por algún método de imputación más complejo. En este caso, se optó por imputar los datos utilizando el método de k-NN (k-Nearest Neighbors).

En la figura 14 se puede notar que la instancia i6 tiene un dato faltante z6, para poder imputar ese dato faltante. El método de k-NN buscará en el espacio de características las instancias u observaciones más cercanas a ese dato faltante, la cercanía de cada observación esta dada por una distancia en específica, en nuestro caso se uso la distancia euclediana. Por último se tiene que determinar el número de vecinos a tomar en cuenta, en este caso se tomo a 4 vecinos con respecto a la observación. Una vez obtenidos los k vecinos más cercanos, el método calcula el promedio de sus valores para imputar el dato faltante

Figura 14

Imputación por k-NN



Nota: Representación gráfica de como se realizó la imputación por k-NN. Elaborado por: (El autor, 2024).

Lo mismo que se mencionó para el ejemplo de la figura 14 se llevó a cabo en la base de datos utilizada en este trabajo, con la variante de que ahora se trabaja en un espacio de 25 caracterís-

ticas. Inicialmente, los datos se presentaban como se muestra en la figura 15 (a). Después de la imputación de los datos categóricos y cuantitativos, la base de datos ahora se visualiza como se muestra en la figura 15 (b), es decir, ya no presenta ningún dato faltante dentro de sus variables.

Figura 15

Imputación a las variables categóricas y cuantitativas de la base de datos

```
> head(Base_de_datos_CKD)
# A tibble: 6 × 25
  age   bp    sg    al    su    rbc    pc       pcc         ba     bgr   bu    sc    sod   pot   hemo  pcv   wc    rc
  <chr> <chr> <chr> <chr> <chr> <chr>  <chr>    <chr>       <chr>  <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
1 48    80    1.020 1     0     ?      normal   notpresent  notpr… 121   36    1.2   ?     ?     15.4  44    7800  5.2
2 7     50    1.020 4     0     ?      normal   notpresent  notpr… ?     18    0.8   ?     ?     11.3  38    6000  ?
3 62    80    1.010 2     3     normal normal   notpresent  notpr… 423   53    1.8   ?     ?     9.6   31    7500  ?
4 48    70    1.005 4     0     normal abnormal present     notpr… 117   56    3.8   111   2.5   11.2  32    6700  3.9
5 51    80    1.010 2     0     normal normal   notpresent  notpr… 106   26    1.4   ?     ?     11.6  35    7300  4.6
6 60    90    1.015 3     0     ?      ?        notpresent  notpr… 74    25    1.1   142   3.2   12.2  39    7800  4.4
```

(a) *Gráfico A*

```
> head(moment)
  age bp   bgr bu  sc   sod  pot hemo pcv   wc  rc    sg al su      rbc       pc        pcc         ba
1  48 80 121.0 36 1.2 138.5 4.95 15.4  44 7800 5.2 1.020  1  0   normal   normal notpresent notpresent
2   7 50 107.5 18 0.8 140.0 4.15 11.3  38 6000 5.3 1.020  4  0 abnormal   normal notpresent notpresent
3  62 80 423.0 53 1.8 135.5 4.60  9.6  31 7500 4.0 1.010  2  3   normal   normal notpresent notpresent
4  48 70 117.0 56 3.8 111.0 2.50 11.2  32 6700 3.9 1.005  4  0   normal abnormal    present notpresent
5  51 80 106.0 26 1.4 141.5 4.00 11.6  35 7300 4.6 1.010  2  0   normal   normal notpresent notpresent
6  60 90  74.0 25 1.1 142.0 3.20 12.2  39 7800 4.4 1.015  3  0 abnormal   normal notpresent notpresent
```

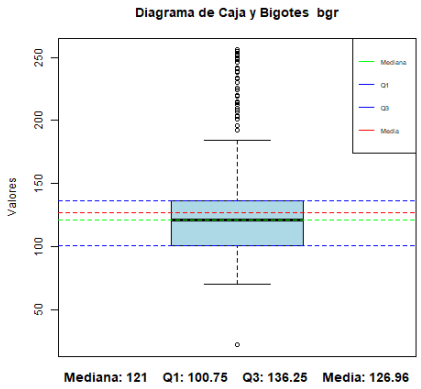
(b) *Gráfico B*

Nota: (a) Base de datos sin imputar (b) Base de datos imputada por k-NN. Elaborado por: (El autor, 2024).

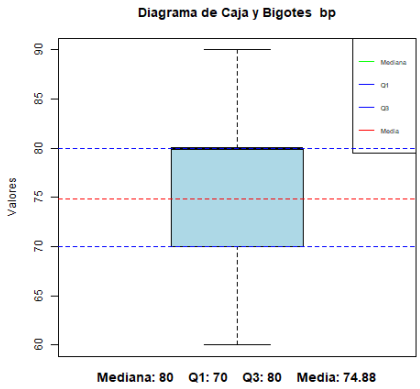
Comparando los diagramas de caja y bigotes de la figura 12, en la cual los datos no fueron tratados respecto a sus outliers ni se aplicó imputación, con el de la Figura 16, donde ya se abordaron los outliers y se realizó la imputación de datos faltantes, se observa una significativa reducción de datos atípicos. Este cambio sugiere que la información ahora es más precisa y refleja con mayor fidelidad las situaciones que podrían surgir en la práctica clínica.

Figura 16

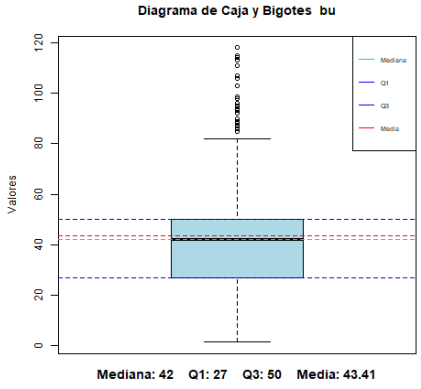
Diagrama de caja y bigotes de las varaiables cuantitativas después del preprocesamiento



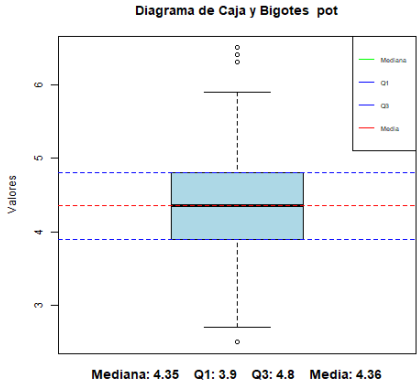
(a) *Glucosa en sangre aleatorio*



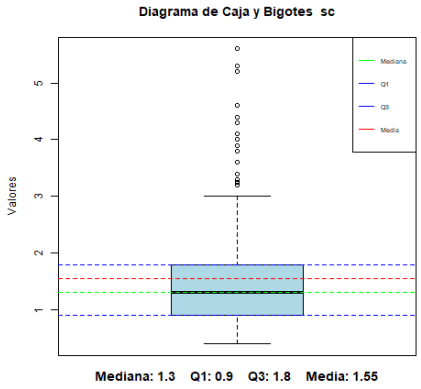
(b) *Presión Arterial*



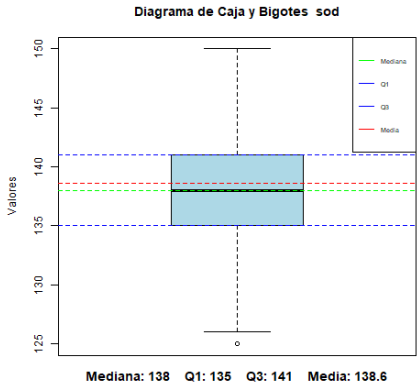
(c) *Urea en sangre*



(d) *Potasio*



(e) *Creatinina sérica*



(f) *Sodio*

Nota: Elaborado por: (El autor, 2024).

5.3. Análisis de los datos

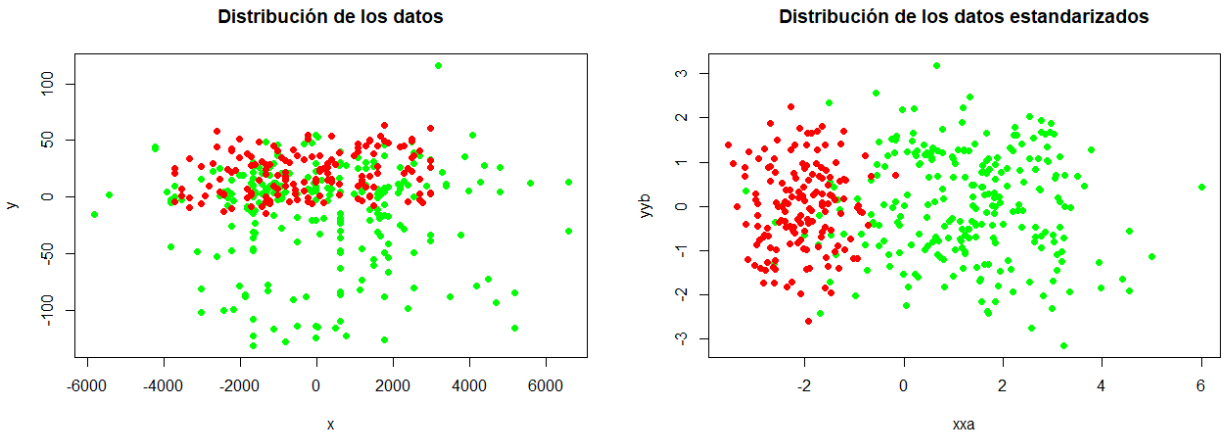
5.3.1. Distribución de los datos

Para poder observar la distribución de los datos en dos dimensiones, se tuvo que aplicar un escalamiento multidimensional a las variables de tipo cuantitativo, con el único fin de que generar dos nuevos factores que tendran la mayoría de la información de toda la base de datos. En la figura 17 (a), se puede observar la distribución de los datos en dos dimensiones, vemos que apenas y se logra diferenciar las clases de personas con ERC y sin ella, esto se debe a que no se aplicó ningun tipo de estandarización y quizá las variables con mayores magnitudes no dejan diferenciar bien algun patrón o grupo que podamos tener en los datos. En el caso de la figura 17 (b), ahora sí se puede diferenciar de forma fácil como se dividen las personas con ERC y sin ella, en este caso se aplicó una estandarización a todas la variables.

Ahora el verdadero problema recae en poder diferenciar o agrupar dentro de las clases de las personas con ERC cada una de las etapas en las que se encuentra cada instancia, esto ya no se lo puede hacer de forma visual, y es aquí donde entran los modelos de ML, son ellos quienes finalmente logran dentro de esos datos poder clasificarlos.

Figura 17

Distribución de los datos en 2D.



(a) *Datos sin escalar* **(b)** *Datos escalados*

Nota: Puntos verdes: Personas con ERC - Puntos Rojos: Persona sin ERC. Elaborado por: (El autor, 2024).

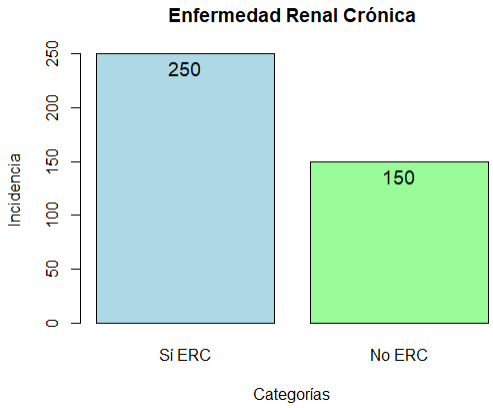
5.3.2. Cantidad de personas con ERC

Es importante que se tenga un balance entre las personas que fueron diagnosticadas con ERC y aquellas que no la padecen. Pero al mismo tiempo, se requiere un mayor porcentaje dentro de las personas con esta enfermedad, ya que más adelante será clave ver sus etapas e incidencia dentro de las mismas.

Según la información de la figura 18, dentro de la base de datos, alrededor del 62.5% de las personas tienen ERC, mientras que el otro 37.5% corresponde a aquellas que no padecen esta enfermedad.

Figura 18

Incidencia de la ERC en la base de datos



Nota: Personas con ERC en la base de datos. Elaborado por: (El autor, 2024).

5.3.3. Incidencia de factores asociados a la ERC

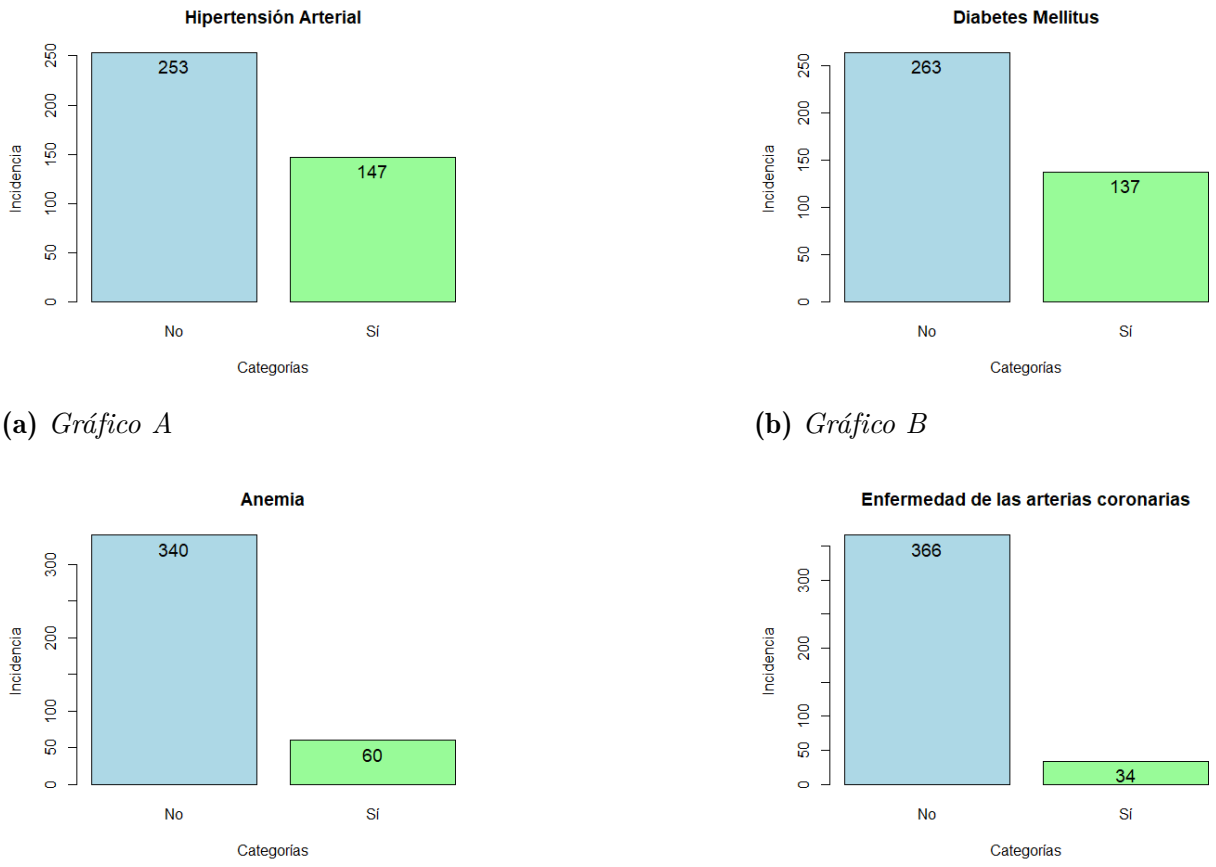
Como se discutió en los fundamentos teóricos, es crucial reconocer los factores de riesgo asociados que pueden contribuir o prolongar la probabilidad de padecer ERC. En la base de datos, como se observa en la figura 19, a pesar de tener un mayor porcentaje de personas con ERC, ocurre lo contrario en el caso de los factores de riesgo. Se registra un menor porcentaje de personas con Hipertensión Arterial: 147 en comparación con 253 sin esa patología. Lo mismo sucede en el caso de personas con Diabetes Mellitus, en la cual se encuentran 263

personas sin diabetes y 137 con ella. También, en el caso de la anemia, se tiene a 60 personas con anemia y a 340 sin ella. Por último, y con una diferencia más significativa, se presentan las personas con enfermedad de las arterias coronarias, ya que se cuenta con tan solo 34 personas con esta patología y 365 personas sin la misma.

Es importante recalcar que al tener uno de los factores de riesgo asociados con la ERC, es solo cuestión de tiempo para adquirir los demás factores y, además, para que la etapa de la ERC avance.

Figura 19

Factores de riesgo dentro de la base de datos



(a) *Gráfico A*

(b) *Gráfico B*

(c) *Gráfico C*

(d) *Gráfico D*

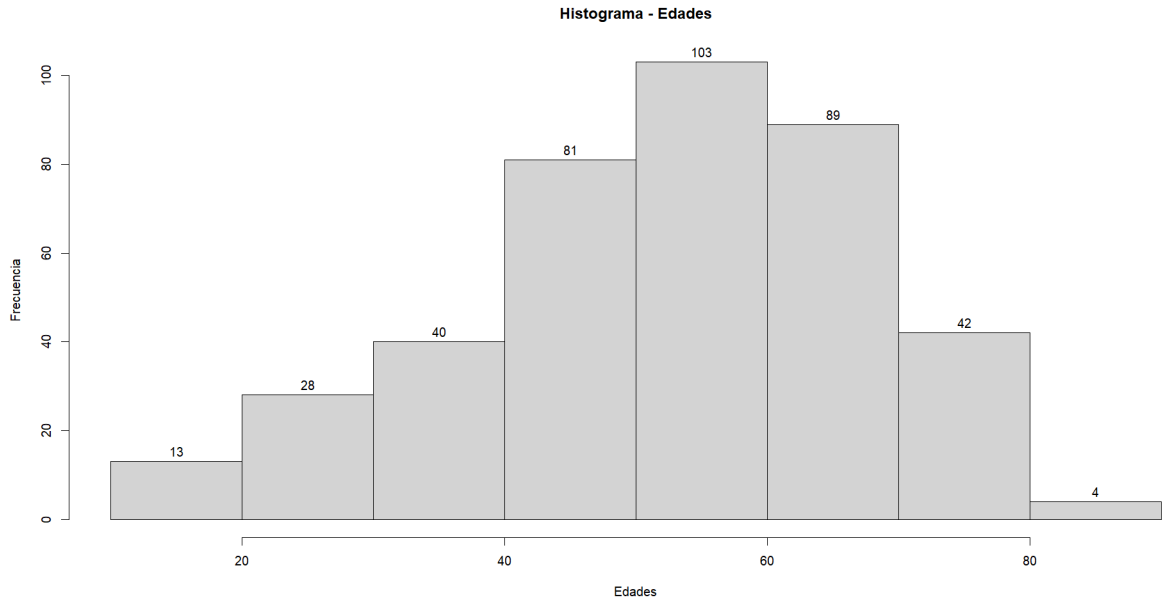
Nota: (a) Incidencia HA (b) Incidencia DM (c) Incidencia Anemia (d) Incidencia ECV. Elaborado por: (El autor, 2024).

5.3.4. Edad de los pacientes

La distribución de los datos en la variable de edad abarca un amplio intervalo que va desde los 11 hasta los 90 años. Aunque es conocido que la ERC es más común en adultos, puede haber casos en los que afecte a niños, causados por malformaciones renales congénitas, trastornos genéticos, entre otros. La mayor concentración de edades en la base de datos se encuentra entre los 30 y los 80 años, y solo unas pocas personas se encuentran en el rango de los 80 a los 90 años, como se indica en la figura 20.

Figura 20

Histograma de la edad de los pacientes



Nota: Se ve que en la edad se tiene un rango que va desde los 11 hasta los 90 años. Elaborado por: (El autor, 2024).

5.4. Procesamiento de la base de datos

5.4.1. Adición de variables

Originalmente, la base de datos contaba con 25 características. Sin embargo, con el fin de cumplir los objetivos de este trabajo, se añadieron las variables que se encuentran en la tabla 6.

Tabla 6

Adición de variables

Variable	Tipo	Unidades/Rango
Genero	Nominal	(1- Mujer, 0- Hombre)
Tasa de filtrado glomerular estimada	Numérica	mL/min/1,73 m2
Etapas	Nominal	(0,1,2,3,4,5)

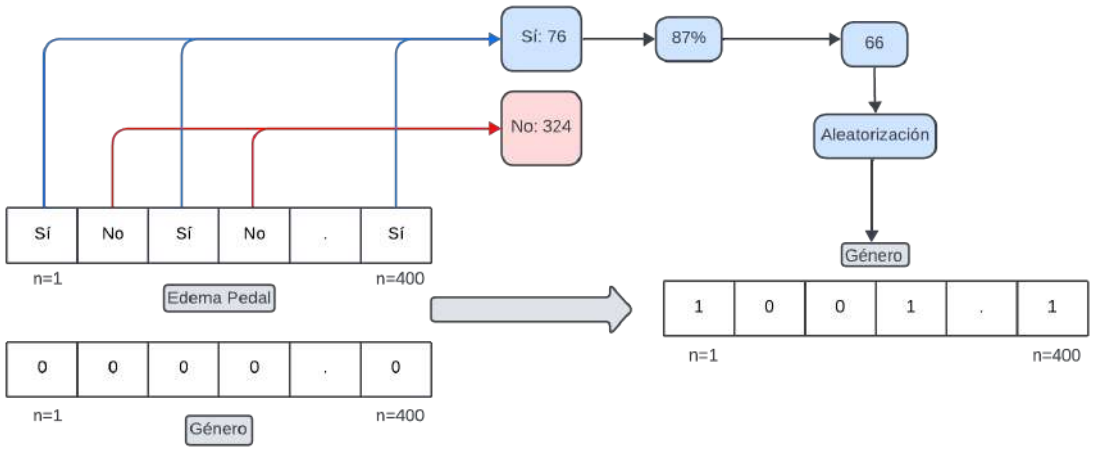
Nota: Estas tres variables se añadieron a la base de datos. Elaborado por: (El autor, 2024).

La base de datos original no contiene información sobre el género de las personas entre sus características, lo cual es de gran importancia, ya que es uno de los parámetros necesarios en la ecuación CKD-EPI. Por esta razón, se recurrió a buscar alguna variable clave dentro de la base de datos que pudiera estimar el género de los pacientes. El parámetro seleccionado fue el edema pedal. Según la literatura, algunos tipos de esta afectación son habituales en mujeres, ya que en este género existe una predisposición hormonal que provoca la aparición de estos edemas, especialmente en épocas de cambios como la menarquia, menopausia o un embarazo (ROYO, 2022) (Douketis, 2022). Incluso se menciona que afecta a las mujeres en un 87% más que a los hombres, especialmente entre la segunda y tercera década de la vida (Villegas, Lazcano, y de Lourdes Lazcano Mendoza, 2014).

Para estimar el género femenino, se siguió el proceso descrito en la figura 21. En primer lugar, se creó un arreglo de ceros del mismo tamaño que la variable edema pedal. A continuación, se determinó la cantidad total de personas con edema pedal, que resultó en 76 personas de las 400 en total. De estas 76 personas con edema pedal, solo se consideró el 87%, lo que representaría el género femenino y dio como resultado 66 personas. Posteriormente, con estos datos listos, se asignó el valor de uno a un conjunto de posiciones correspondientes al 87% de las instancias donde la variable edema pedal indicaba 'sí'. Las demás posiciones dentro de la variable de género permanecieron en cero. Finalmente, esta nueva variable se adicionó a la base de datos, representando el género femenino.

Figura 21

Adición de variables nuevas en la base de datos



Nota: Se contó con un total de 324 personas que no presentaban edema pedal, en comparación con 76 que sí padecían esta afección. Elaborado por: (El autor, 2024).

Como ya se hizo la adición de la variable de género dentro de la base de datos, ahora ya es posible calcular la TFGe. Para esto se hizo uso de la ecuación CKD-EPI (A. S. Levey y cols., 2009), esta ecuación se representa en la figura 22.

Figura 22

Ecuación CKD-EPI para el cálculo de la TFGe.

CKD-EPI equation (white or other subjects)[11]

If female

If Scr ≤ 0.7 mg/dl

GFR (ml/min per 1.73 m^2) = 144 × $(Scr/0.7)^{-0.329}$ × $(0.993)^{age}$

If Scr > 0.7 mg/dl

GFR (ml/min per 1.73 m^2) = 144 × $(Scr/0.7)^{-1.209}$ × $(0.993)^{age}$

If male

If Scr ≤ 0.9 mg/dl

GFR (ml/min per 1.73 m^2) = 141 × $(Scr/0.9)^{-0.411}$ × $(0.993)^{age}$

If Scr > 0.9 mg/dl

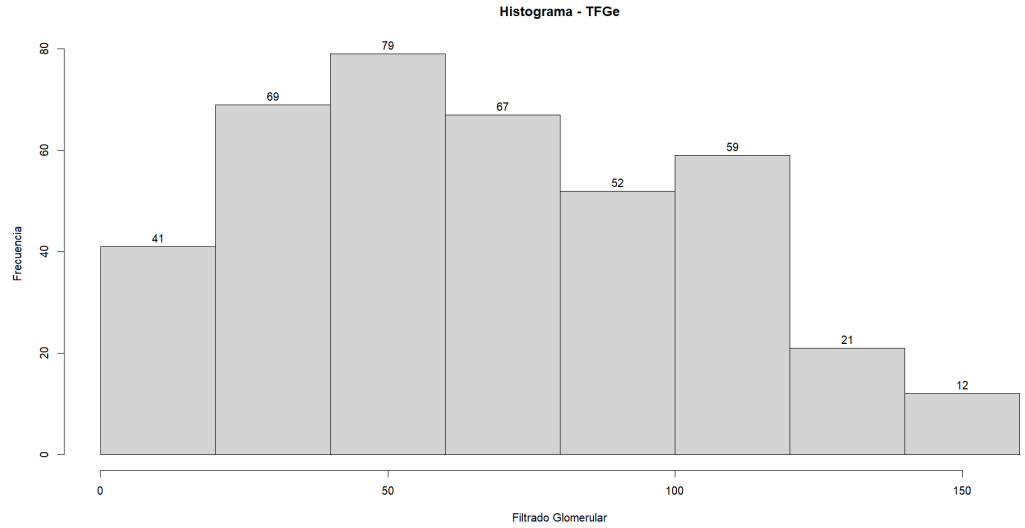
GFR (ml/min per 1.73 m^2) = 141 × $(Scr/0.9)^{-1.209}$ × $(0.993)^{age}$

Nota: Esta ecuación es superior a las ecuaciones de MDRD existentes en términos de estimación de la TFG. Fuente: (Liao, Liao, Liu, Xu, y Zeng, 2011).

Después de aplicar la ecuación utilizando las características de edad y creatinina sérica, se puede observar en la figura 23 una amplia distribución de los valores de la TFGe, lo cual es positivo, ya que esta variable será crucial para la clasificación de las etapas de la ERC. En la tabla 7, se aprecia que la mayoría de los datos se encuentran en el rango de 67 mL/min/1,73 m^2, con una desviación estándar de aproximadamente 36.42 mL/min/1,73 m^2 y un rango que va desde 7 hasta 157 mL/min/1,73 m^2. Es importante destacar que estos valores de TFGe abarcan a personas con ERC desde la etapa más grave hasta las etapas iniciales.

Figura 23

Histograma de la TFGe.



Nota: Elaborado por: (El autor, 2024).

Tabla 7

Medidas de tendencia central y dispersión de la TFGe

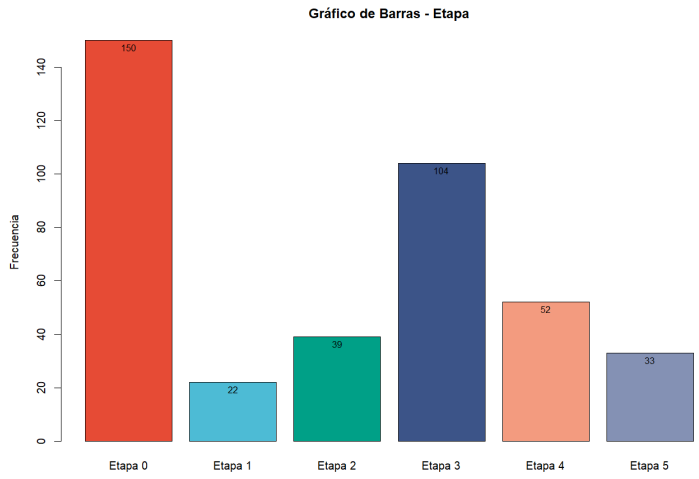
Variable	Media	Mediana	Desviación Estándar	Min	Max
TFGe	67.075	64.079	36.42	7.56	157.16

Nota: Medidas de tendencia central y dispersión de la TFGe. Elaborado por: (El autor, 2024).

Una vez obtenida la TFGe, se procedió a clasificar las 250 instancias que presentaban un diagnóstico de ERC en cada una de las etapas correspondientes, que se abordaron en la tabla 1 en el marco teórico. De esta manera, surgió la nueva variable denominada etapa que se encuentra en la figura 24. En la etapa 0 tenemos al 37.5 % de los datos y representan a las personas sin ERC, en cambio el restante son personas con ERC que van desde la etapa 1 hasta la 5, con un porcentaje del 5.5 %, 9.75 %, 26 %, 13 % y el 8.25 % respectivamente.

Figura 24

Gráfico de barras de las etapas



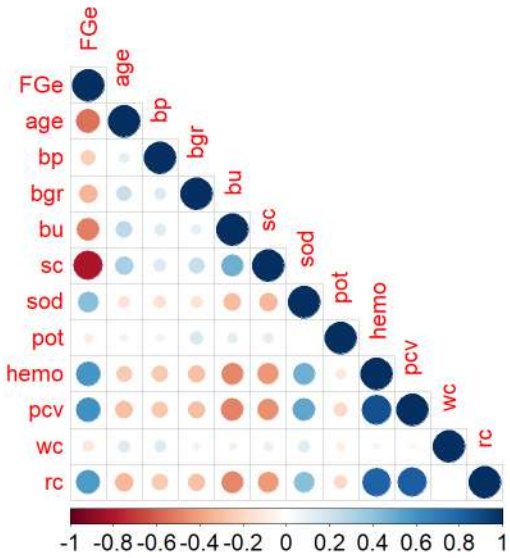
Nota: División de las 400 instancias por etapas en base al nivel de TFGe. Elaborado por: (El autor, 2024).

5.5. Selección de características

Para poder hacer la selección de características en una base de datos con múltiples variables existen varios criterios. Uno de ellos es ver la correlación entre las variables predictoras y la variable a predecir. Por ejemplo en la figura 25 se tiene a la correlación entre pares de todas las variables cuantitativas. Dado que etapa, la variable a predecir es de tipo categórica, no se podía ver el nivel de correlación que tenia esta con las demas variables. Por ese motivo una alternativa fue determinar el nivel de correlación que tenía la TFGe con las demás variables, esto fue posible gracias a que la TFGe esta estrechamente relacionada con la variable etapa.

Figura 25

Correlación de variables cuantitativas



Nota: En la diagonal principal se tiene una correlación de 1 debido a que se trata con su misma variable. Elaborado por: (El autor, 2024).

En la Tabla 8, ya se tiene a los valores de correlación de la TFGe con las demás variables y se evidencia la presencia de variables con niveles de correlación tanto positivos como negativos. La creatinina sérica destaca como la variable más fuertemente correlacionada de manera negativa con un valor de -0.81; esto indica que a medida que el nivel de TFGe aumenta, la

cantidad de creatinina sérica disminuye. En otras palabras, la creatinina sérica muestra una relación inversamente proporcional con la TFGe. Un patrón similar se observa con la edad, aunque con un nivel de correlación negativa más moderado.

Por otro lado, en el caso de las correlaciones positivas, se destaca que el volumen de células empaquetadas presenta una correlación de 0.60. Esto sugiere que a medida que aumenta el volumen de células empaquetadas, la TFGe también tiende a aumentar. Este patrón positivo se repite con la hemoglobina y las células rojas, indicando que un incremento en estos parámetros está asociado con un aumento en la TFGe.

Tabla 8

Variables mas correlacionadas con la TFGe

Variable 1	Variable 2	Coeficiente de correlación
TFGe	Edad (E)	-0.55
	Presión Arterial (PA)	-0.23
	Glucosa en sangre aleatorio (GSA)	-0.33
	Urea en sangre (US)	-0.51
	Creatinina sérica (CS)	-0.81
	Sodio (S)	-0.41
	Potasio (P)	-0.09
	Hemoglobina (H)	0.58
	Volumen de células empaquetada (VCE)	0.60
	Células blancas (CB)	-0.13
	Células rojas (CR)	0.56

Nota: Coeficiente de correlación entre variable 1 y variable 2. Elaborado por: (El autor, 2024).

Después de analizar los valores de correlación, es crucial determinar las variables que se utilizarán en los modelos de ML. Con este propósito, se han propuesto variantes de características que se detallan en la Tabla 9. Se realizarán modelos con esas tres variantes y se escogerá al que tenga la menor cantidad de variables pero que a su vez tengo buenas métricas de clasificación.

Tabla 9

Variantes en la selección de características

Variante	Condición	Características seleccionadas
1	–	Todas las variables.
2	$>= 0.5$	E, US, CS, H, VCE y CR.
3	$>= 0.6$	CS y VCE
4	$>= 0.7$	CS

Nota: Tres variantes de grupos para la selección de características. Elaborado por: (El autor, 2024).

6. Modelado

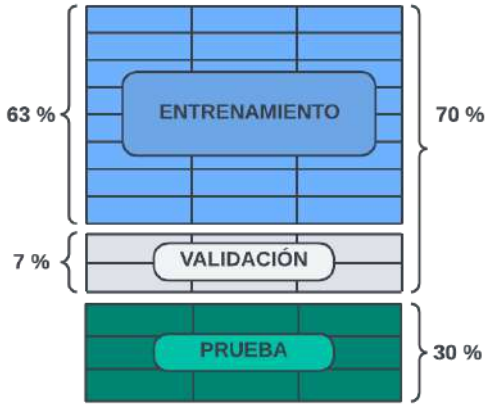
6.1. Aleatorización y conjunto de entrenamiento y prueba

Debido a que en las 250 primeras instancias de la base de datos se encontraban a pacientes con ERC de distintas etapas y en el resto a instancias sin ERC, se decidió realizar una mezcla de los datos en base a un punto semilla, el punto semilla sirve para que se pueda hacer una aleatorización que a futuro pueda ser reproducible.

En la figura 26, se dividió la base de datos ya randomizada en un cojunto de entrenamiento y en un conjunto de prueba, dentro del cojunto de entrenamiento se opto por tomar al 70 % y para prueba al 30 % de la base de datos. De esta forma, queda por etapa la cantidad de instancias que se representan en la tabla 10. Dentro del porcentaje de entrenamiento se tiene a otro porcentaje de datos que servirán para poder aplicar uno o más métodos de validación.

Figura 26

División del conjunto de datos



Nota: Cojunto de entrenamiento: 70 % - Conjunto de prueba: 30 %. Elaborado por: (El autor, 2024).

Tabla 10

Cantidad de datos y etapas en la partición de entrenamiento.

Etapa	Instancias Entrenamiento	Instancias Prueba
0	105	45
1	17	5
2	30	9
3	69	35
4	33	19
5	26	7
Total	280	120

Nota: Existe una mayor prevalencia de la etapa 0 y 3, por lo que será las clases que mejor clasificará. Elaborado por: (El autor, 2024).

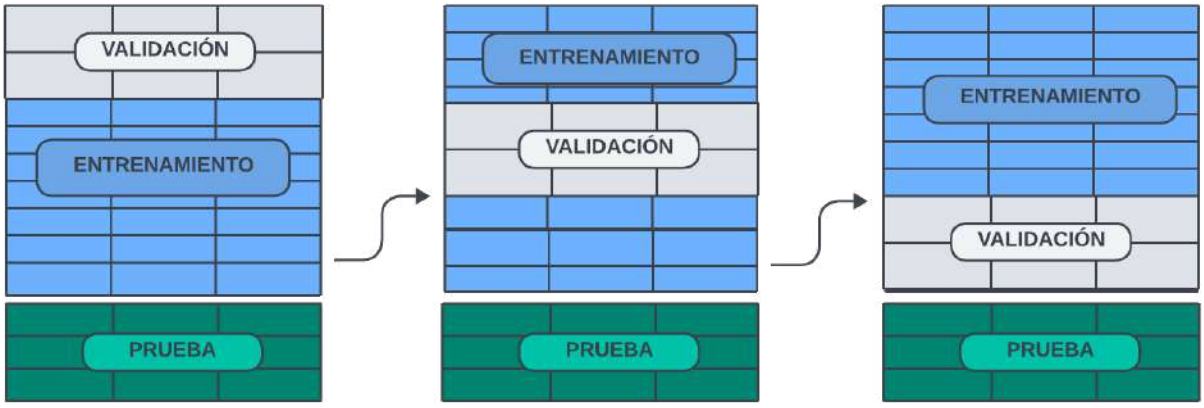
6.2. Método de validación

Para la validación se consideró usar el método de k-fold cross validation dado que es un método robusto para poder identificar rápidamente si un modelo tiende a sobrajustarse a los datos de entrenamiento o en caso contrario, si un modelo esta infraajustado. De igual forma nos permitió ajustar los hiperparámetros de los modelos para saber cuales son los que causan una mejora en sus rendimientos.

En la figura 27 se puede ver como la validación es una partición o también denominada carpeta dinámica, y que la cantidad de carpetas a usar dependerá del parámetro k, por lo general para bases de datos no muy grandes se suele usar un k entre 5 y 10. En este caso se uso un k de 10, es decir 10 carpetas para validar los datos de 10 modelos de ML. Para la validación en cada carpeta se trabajo con el 7% de los datos para validación, el otro 63% se uso para entrenamiento.

Figura 27

Método de validación K-fold cross validation



Nota: El conjunto de validación varía dentro del conjunto de entrenamiento. Elaborado por: (El autor, 2024).

6.3. Modelos de Máquinas de Soporte Vectorial (MSV)

Para los primeros modelos de SVM se implementaron utilizando la libreria e1071 en R. Esta librería permite manejar técnicas de aprendizaje automático, como por ejemplo a

las SVM, al igual que nos ayudan a manejar nuestros propios hiperparámetros para poder encontrar mejores resultados.

Se ajustaron los hiperparámetros de todos los modelos de SVM según la información proporcionada en la tabla 11. En total, se exploraron cuatro valores de costo con el objetivo de determinar en cuál de ellos el modelo mostraba un rendimiento óptimo. Asi mismo, se evaluaron cuatro kernels distintos para identificar la función que mejor se adaptaba a la tarea de clasificación.

Tabla 11

Costos para los modelos de SVM.

Costo	Valor	Kernel
1	1	
2	10	Lineal - Radial
3	100	Polinomial- Sigmoide
4	1000	

Nota: Tres variantes de grupos para la selección de características. Elaborado por: (El autor, 2024).

6.3.1. Primer Modelo SVM con todas las variables

Como primer modelo se consideró tomar a todas las variables, tanto cuantitativas como cualitativas, con el fin de observar el rendimiento al trabajar con toda la información que las características pueden ofrecer.

Primero se aplicó el método de k-fold cross validation con el 70 % de los datos de entrenamiento para poder determinar que hiperparámetros generaban un mejor accuracy, para luego poder evaluar el modelo con la partición de prueba que corresponde al 30 % restante de los datos.. En la tabla 12, se presentan las métricas de accuracy correspondientes al uso de diferentes kernels. Al emplear un kernel lineal con un costo de 10, se obtuvó un accuracy del 68 %. En contraste, al utilizar otros kernels como el polinomial, radial y sigmoidal generaban un acuracy por debajo o igual al 61 % y además estos generaban una gran cantidad

de vectores de soporte, lo cual es malo dado que se puede generar un sobre ajuste en el modelo.

En consecuencia, el modelo más eficaz en este contexto fue el kernel lineal con un costo de 10, dado que no requería que el costo aumente, como se ve en la figura 28, mientras el costo aumentaba el accuracy disminuía ligeramente. Por ejemplo, con un costo máximo de 1000, se observaba una pérdida de 0.004 puntos porcentuales. Aunque esto pueda parecer pequeño, en términos de clasificaciones correctas, se aprecian diferencias significativas.

Tabla 12

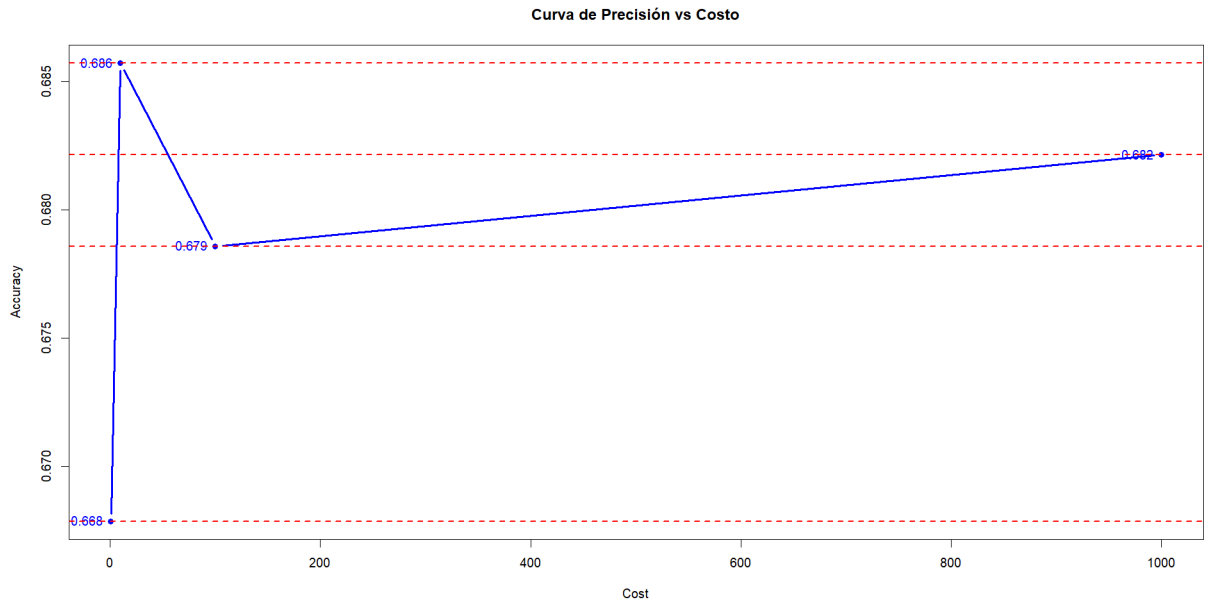
Kernel - Primer modelo.

Kernel	Costo	Vectores de Soporte	Mejor Accuracy
Lineal	10	155	0.68
Polinomial	1000	201	0.60
Radial	10	197	0.61
Sigmoidal	10	165	0.61

Nota: El kernel lineal con un costo de 10 son los mejores hiperparámetros para un modelo con todas las variables. Elaborado por: (El autor, 2024).

Figura 28

Precisión vs Costo - Kernel: lineal - Costo: 10 - MSV



Nota: El accuracy máximo es de 0.685. Elaborado por: (El autor, 2024).

En la figura 29 (a), se presenta la matriz de confusión evaluada utilizando los mismos datos de entrenamiento que se utilizaron para ajustar el modelo con los hiperparámetros óptimos. El modelo tan solo se equivoca en 7 instancias. Por ejemplo, a cuatro instancias las está clasificando en la etapa 3, pero en realidad, una le corresponde en la etapa 2 y las demás en la 5. De igual manera, en la etapa 5, está clasificando incorrectamente 2 instancias, dado que estas corresponden a las etapas 2 y 4, respectivamente. Por último, se equivoca en una instancia en la etapa 4, ya que en realidad esa corresponde a la etapa 5. En el caso de las etapas 1 y 2 no se llegó a equivocar en ninguna instancia.

Figura 29

Matriz de confusión Primer Modelo MSV

(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion entrenamiento: 279 clasificaciones correctas - Matriz de confusión prueba: 88 clasificaciones correctas. Elaborado por: (El autor, 2024).

En la Figura 29 (b), se evaluó el modelo utilizando el 30 % de los datos de prueba para verificar su desempeño al enfrentarse a datos que no había visto anteriormente, es decir, con datos nuevos. Además, en la Tabla 14 se presentan todas las métricas de evaluación correspondientes a un modelo clasificador. Ahora se equivoca en 32 instancias de 120, por lo cual esta clasificando bien un total de 88 instancias, es por esto que tiene un accuracy de todo el modelo del 0.73 %.

Al examinar la métrica de precisión, se destaca que, en la etapa 5, solo el 16 % de las instancias clasificadas como positivas fueron identificadas correctamente. Esta observación se confirma al consultar la tabla 13, donde notamos que la etapa 5 presenta la mayor cantidad de falsos positivos, totalizando 16, en comparación con los verdaderos positivos, que son solo 3, esto pone en evidencia que el modelo enfrenta dificultades al intentar clasificar con precisión las instancias específicas de la etapa 5, algo similiar ocurre con la etapa 4 dado que tiene un accuracy del 50 %.

En cuanto a la sensibilidad del modelo, se destaca una baja sensibilidad en las etapas desde la 1 hasta la 5. Este fenómeno se atribuye a que son las etapas con la mayor tasa de falsos negativos, en la tabla 13 vemos que se tiene 3, 2, 10, 12 y 4 falsos negativos respectivamente, la que tiene una mayor cantidad de estos es la de la etapa 4, es por esto que tiene una sensibilidad de 0.37 siendo esta la más baja.

Por último en temas de especificidad, la mayoria de las etapas tienen esta métrica en un

valor por encima del 80%, esto significa que el modelo tiene una buena capacidad para evitar tener una gran cantidad de falsos positivos por etapa.

Tabla 13

Matriz de observación - Primer modelo.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	44	73	2	1
Etapa 1	2	114	1	3
Etapa 2	7	107	4	2
Etapa 3	25	83	2	10
Etapa 4	7	94	7	12
Etapa 5	3	97	16	4

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 14

Métrica de evaluación - Primer modelo.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	1.00	0.97	1.00	0.97	1.00	0.95
Precisión	1.00	0.96	1.00	0.67	1.00	0.64
Medida F	1.00	0.97	1.00	0.50	1.00	0.70
Sensibilidad	1.00	0.98	1.00	0.40	1.00	0.78
Especificidad	1.00	0.97	1.00	0.99	1.00	0.96
	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	0.99	0.90	1.00	0.84	0.99	0.83
Precisión	1.00	0.92	1.00	0.50	0.92	0.16
Medida F	0.99	0.81	1.00	0.42	0.98	0.23
Sensibilidad	0.98	0.71	1.00	0.37	1.00	0.43
Especificidad	1.00	0.98	1.00	0.93	0.99	0.86
Accuracy General					0.99	0.73
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

6.3.2. Segundo Modelo SVM con primer variante

Como segundo modelo, se consideró usar tan solo las variables que tenian el 50 % o más de correlación con la variable TFGe. Estas incluyeron la edad, úrea en sangre, creatinina sérica, hemoglobina, volumen de células empaquetadas y la cantidad de células rojas. Reduciendo en total de 27 variables dependientes a tan solo 6 variables dependientes.

Utilizando el método de k-fold cross validation, se identificaron los hiperparámetros que resultaron en un mejor accuracy. Los resultados de la tabla 15 indican que tanto el uso de un kernel lineal como uno polinomial con costos diferentes generan un nivel de precisión del 75 %, marcando una mejora del 7 % en comparación con el primer modelo que incluía todas las variables.

Diversos factores pueden explicar este aumento de precisión. En primer lugar, al incrementar el número de variables en el modelo, se corre el riesgo de generar una estructura demasiado compleja, propensa al sobreajuste de los datos de entrenamiento. Además, el aumento en la dimensionalidad del espacio de características y la dispersión de los datos en esa dimensión podrían estar dificultando la clasificación de las etapas. En conjunto, estos aspectos plantean desafíos significativos para el proceso de clasificación.

Para este modelo SVM, se decidió utilizar nuevamente un kernel lineal, ya que se lograba el mejor accuracy con un costo de 1, a diferencia de otros kernels que requerían un costo más elevado, alrededor de 10 o 100, para alcanzar un rendimiento cercano al del lineal, siempre se econtraban por debajo de 0.3 % o 0.2 % puntos porcentuales. La Figura 30 ilustra cómo el costo del modelo con kernel lineal disminuye al aumentar su valor, por lo que es suficiente utilizar un costo tan bajo como 1 para lograr la optimización deseada.

Tabla 15

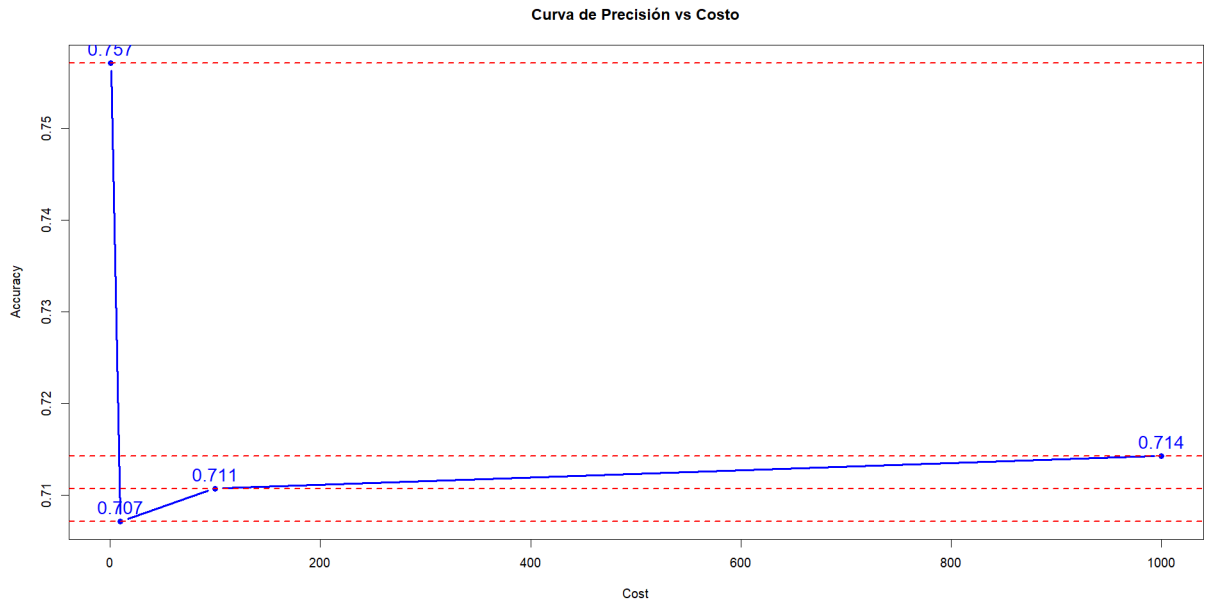
Kernel - Segundo modelo.

Kernel	Costo		Mejor Accuracy
Lineal	1	162	0.75
Polinomial	100	153	0.72
Radial	10	173	0.73
Sigmoidal	1	188	0.66

Nota: Solo el kernel lineal con un costo de 1 genera un buen modelo. Elaborado por: (El autor, 2024).

Figura 30

Precisión vs Costo - Kernel: lineal - Costo: 1 - MSV

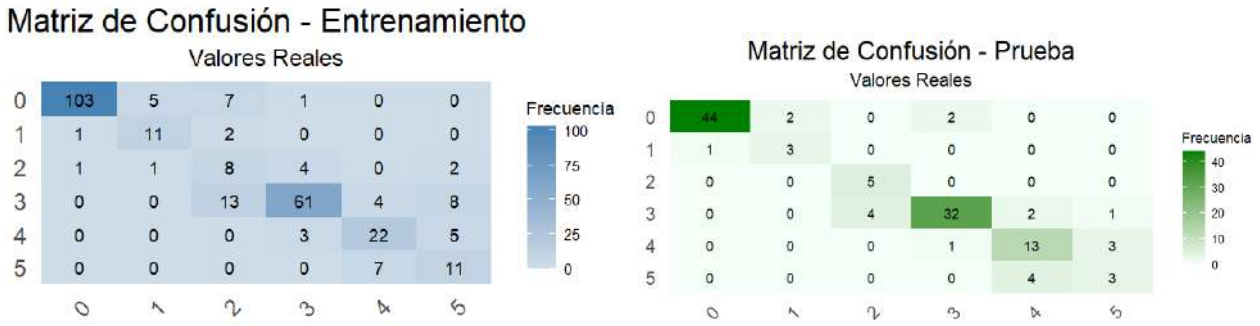


Nota: El accuracy máximo es de 0.757 pero a medida que aumentamos el costo lo reducimos. Elaborado por: (El autor, 2024).

Como se evidencia en la Figura 31 (a), se observa una disminución en la cantidad de aciertos por instancia, y ahora se están cometiendo errores en las etapas 0 y 1. Anteriormente, durante el entrenamiento con todas las variables, el modelo no presentaba errores al validar. Un aspecto destacado en la matriz de confusión es la identificación incorrecta de un total de 13 personas como si no tuvieran alguna etapa de ERC, cuando en realidad 5 pertenecen a la etapa 1, 7 a la 2 y 1 a la 3. Estos casos se clasifican como falsos positivos en relación con un diagnóstico positivo de ERC. Es crucial evitar este tipo de errores, ya que, en el caso de personas con ERC no tratadas, la supervivencia de los pacientes estaría en riesgo.

Figura 31

Matriz de confusión Segundo Modelo MSV



(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion entrenamiento: 216 clasificaciones correctas - Matriz de confusión prueba: 100 clasificaciones correctas. Elaborado por: (El autor, 2024).

Cuando se probó el mejor modelo con la partición de prueba se obtuvo la matriz de confusión de la figura 31 (b) y de igual forma la matriz de observación de la tabla 16 y las métricas de evaluación de la tabla 17. Este modelo ahora se equivoca en 20 instancias de 120, por lo cual acierta un total de 100 instancias, es por esto que tiene un accuracy general mayor al anterior modelo con un valor de 83%.

En el caso de la métrica de precisión, se destaca que la etapa 1, 4 y 5 son las que tienen menor cantidad de instancias positivas clasificadas correctamenete, es por esto que tienen un valor por debajo del 80% y en el caso de la etapa 2 tiene una precisión del 100%, pero esto no me garantiza que el modelo nunca se va equivocar en esa etapa. De hecho una precisión del 100% no me garantiza que no haya falsos negativos en esa etapa, esto es algo que la precisión no toma en cuenta, justamente este es el caso dado que en la etapa 2 tenemos a 4 instancias que en realidad pertenecían a esta etapa.

En la sensibilidad del modelo, se identifica un valor bajo en la etapa 1, 2, 4 y 5, es por esto que tienen un valor del 60%, 55%, 68% y 43% respectivamente, siendo la etapa 5 la que tiene un porcentaje más bajo dado que posee una menor cantidad de datos en la partición de prueba. En el caso de las etapas 0 y 3, estas tienen una sensibilidad por encima del 90% lo cual nos quiere decir que tienen un bajo número de falsos negativos en relación a cada etapa.

Finalmente, al examinar la especificidad, se observa que la mayoría de las etapas presentan un valor superior al 90 %. Esto indica que el modelo posee una sólida capacidad para evitar la clasificación errónea de un gran número de falsos positivos por etapa. En particular, en la etapa 2, la especificidad alcanza el 100 %, ya que el modelo no clasifica erróneamente otras etapas como si pertenecieran a la etapa 2.

Tabla 16

Matriz de observación - Segundo modelo.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	44	71	4	1
Etapa 1	3	114	1	2
Etapa 2	5	111	0	4
Etapa 3	32	78	7	3
Etapa 4	13	97	4	6
Etapa 5	3	109	4	4

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 17

Métrica de evaluación - Segundo modelo.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	0.95	0.96	0.97	0.97	0.89	0.97
Precisión	0.89	0.92	0.78	0.75	0.5	1.00
Medida F	0.93	0.95	0.71	0.67	0.35	0.71
Sensibilidad	0.98	0.98	0.65	0.60	0.27	0.55
Especificidad	0.92	0.95	0.99	0.99	0.97	1.00
	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	0.88	0.92	0.93	0.92	0.92	0.93
Precisión	0.71	0.82	0.73	0.76	0.61	0.43
Medida F	0.79	0.86	0.70	0.72	0.50	0.43
Sensibilidad	0.88	0.91	0.67	0.68	0.42	0.43
Especificidad	0.88	0.92	0.97	0.96	0.97	0.96
Accuracy General					0.77	0.83
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

6.3.3. Tercer Modelo SVM con segunda variante

Para el tercer modelo se tomaron solo dos variables y en este caso eran las variables que cumplian con un nivel de correlación mayor o igual al 0.6 con respecto a la TFGe, estas fueron la creatinina sérica y el volumen de células empaquetadas.

Para poder identificar los mejores hiperparámetros para el modelo con dos variables se

aplicó nuevamente el método de k-fold cross validation. En la tabla 18 se puede identificar que tanto los tres primeros kernels generaban un accuracy del 77% y a su vez cada uno generaba un costo diferente con un número de vectores de soporte diferente. Se tomó en cuenta como mejores hiperparámetros a el uso de un kernel radial con un costo de 100, dado que tenía un accuracy relativamente alto con un valor del 77% y de igual forma tenía 144 vectores de soporte a diferencia del lineal con 157 vectores o el polinomial con 139 vectores, pero como punto negativo un costo alto de 1000. Para el costo del modelo no se consideró usar un costo alto debido a que podía aumentar el número de vectores de soporte, como se observa en la figura 32 el máximo accuracy del modelo se encuentra al usar un costo de 100 y a medida que el costo aumentaba a 1000 el accuracy del modelo se reducía.

Tabla 18

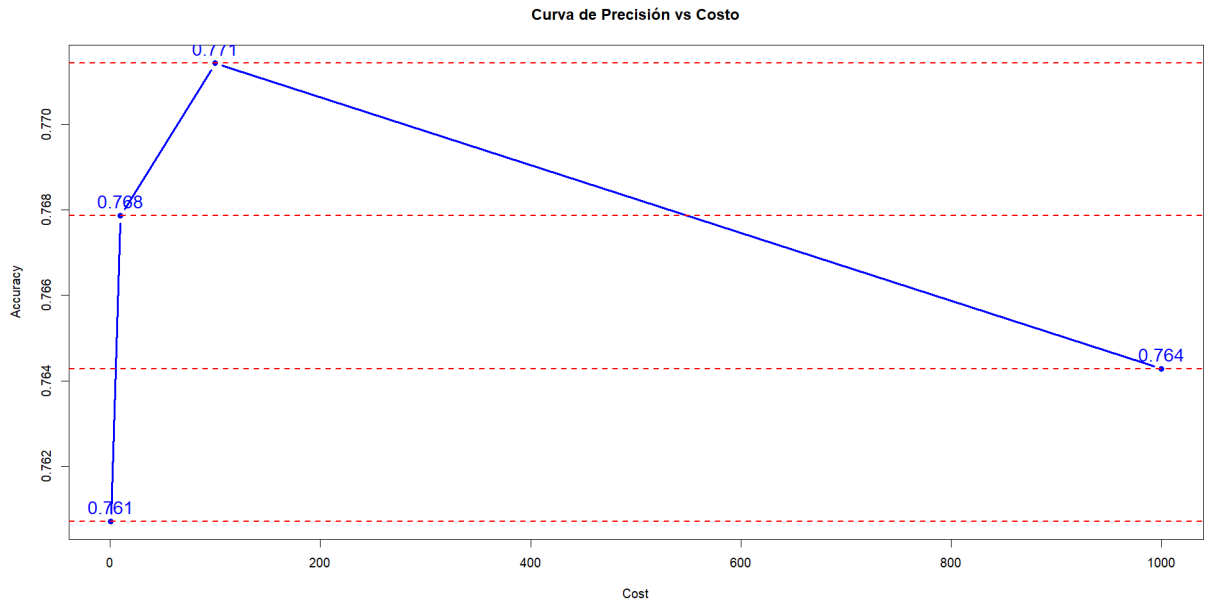
Kernel - Tercer modelo.

Kernel	Costo	Vectores de Soporte	Mejor Accuracy
Lineal	100	157	0.77
Polinomial	1000	139	0.77
Radial	100	144	0.77
Sigmoidal	1	189	0.59

Nota: Tanto el kernel lineal, polinomial y radial generan un buen modelo. Elaborado por: (El autor, 2024).

Figura 32

Precisión vs Costo - Kernel: Radial - Costo: 100 - MSV



Nota: El Accuracy máximo es de 0.771. Elaborado por: (El autor, 2024).

Después de haber entrenado el modelo con esos hiperparámetros, se probó con los datos de entrenamiento para evaluar su comportamiento en la clasificación. Se observa en la matriz de confusión de la figura 33 (a) que, a pesar de haber utilizado la partición de entrenamiento, el modelo comete errores en cada una de las etapas. Por ejemplo, en las etapas 1, 2 y 5, se redujo significativamente la cantidad de aciertos.

En la etapa 1, el modelo clasificó correctamente a 10 personas, pero cometió errores en 7 instancias, a su vez clasificó incorrectamente a 6 personas como si no tuvieran ERC cuando, en realidad, pertenecían a esta etapa. Además, clasificó a 1 persona en la etapa 2, cuando en realidad pertenecía a la etapa 1. En cambio en la etapa 2 se equivoca en 26 instancias y dentro de esas toma a 9 instancias como si no tuvieran ERC, 1 para la etapa 1 y 4 para la 3. En el caso de la etapa 5, se esta equivocando en la mitad de los casos. Con tan solo ver este comportamiento ya vemos que el modelo comete errores graves al tener muchos casos de falsos negativos, por lo que en la etapa de prueba almenos esas tres etapas se veran perjudicadas.

Figura 33

Matriz de confusión Tercer Modelo MSV

(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion entrenamiento: 228 clasificaciones correctas - Matriz de confusión prueba: 98 clasificaciones correctas. Elaborado por: (El autor, 2024).

En la figura 33 (b) se evaluó el modelo con el 30 % de los datos de prueba, en la tabla 19 se encuentra la matriz de observación y en la tabla 20 las métricas de evaluación del modelo. Por ejemplo ahora vemos que se equivoca en 22 instancias de 120, por lo cual esta clasificando bien un total de 98 instancias, es por esto que tiene un accuracy de todo el modelo del 0.81 %.

En cuanto a la precisión las únicas etapas en las que el modelo es preciso es en la 0 y en la 5 dado que tiene un valor de 95 % y 100 % respectivamente. En las otras etapas el valor no es tan bajo pero se encuentran por debajo del 80 %, siendo la etapa 2 la que más baja precisión tiene dado que tiene un valor del 50 %, es por esto que se considera a esta la que más dificultad tiene el modelo en clasificar.

En la sensibilidad del modelo, se identifica un valor bajo en la etapa 1, 2, 5, con un valor del 40 %, 11 % y 71 % respectivamente, siendo la etapa 5 la que tiene un porcentaje más bajo dado que posee una menor cantidad de datos en la partición de prueba. En el caso de las etapas 0, 3 y 4, estas tienen una sensibilidad por encima del 80 % lo cual nos quiere decir que tienen un bajo número de falsos negativos en relación a cada etapa.

Finalmente, al examinar la especificidad, observamos que la mayoría de las etapas presentan un valor superior al 90 %. Esto indica que el modelo posee una sólida capacidad para evitar la clasificación errónea de un gran número de falsos positivos por etapa. En particular, en la etapa 5, la especificidad alcanza el 100 %, ya que el modelo no clasifica erróneamente otras

etapas como si pertenecieran a la etapa 5.

Tabla 19

Matriz de observación - Tercer modelo.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	45	70	5	0
Etapa 1	2	114	1	3
Etapa 2	1	111	0	8
Etapa 3	29	74	11	6
Etapa 4	16	96	5	3
Etapa 5	5	113	0	2

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 20

Métrica de evaluación - Tercer modelo.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	0.95	0.95	0.97	0.97	0.90	0.92
Precisión	0.87	0.90	1.00	0.67	0.55	0.50
Medida F	0.93	0.94	0.74	0.50	0.26	0.18
Sensibilidad	1.00	0.98	0.59	0.40	0.17	0.11
Especificidad	0.91	0.93	1.00	0.99	0.98	0.99
	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	0.89	0.86	0.97	0.93	0.95	0.98
Precisión	0.70	0.72	0.86	0.76	1.00	1.00
Medida F	0.81	0.77	0.88	0.80	0.63	0.83
Sensibilidad	0.96	0.83	0.91	0.84	0.46	0.71
Especificidad	0.87	0.87	0.98	0.95	1.00	1.00
Accuracy General					0.81	0.81
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

6.3.4. Cuarto Modelo SVM con tercera variante

Como último modelo se considero a las variables que tengan un nivel de significancia por encima o igual al 70 %, la única variable que cumple esta condición es la creatinina sérica. Al aplicar el método de validación se obtuvo los datos de la tabla 21. Como mejores hiperparámetros se encuentra un accuracy del 73 %, esto se producía al usar un kernel radial junto con un costo de 1, de igual forma estos hiperparámetros usaban un total de 176 vectores

de soporte. Los demás hiperparámetros tenían como punto negativo el uso de un costo alto, varios vectores de soporte o un accuracy bajo. No se considero un costo más alto con el kernel radial dado que como se observa en la figura 34 al aumentar el costo a 10, 100 o 1000 el accuracy bajaba alrededor de un punto porcentual.

Tabla 21

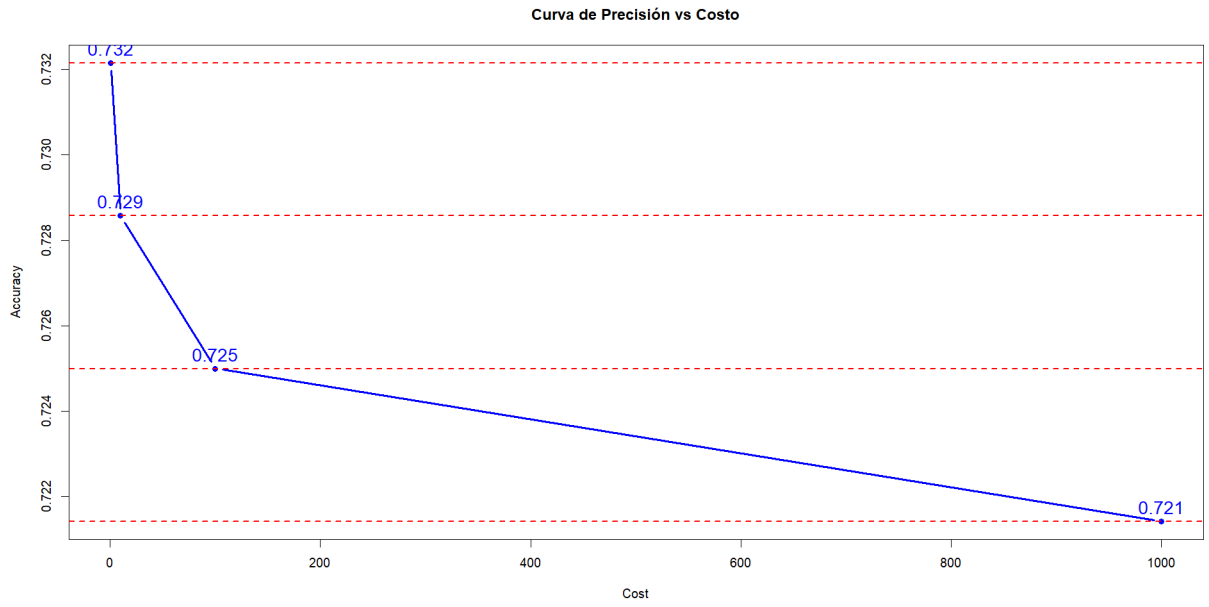
Kernel - Cuarto modelo.

Kernel	Costo	Vectores de soporte	Mejor Accuracy
Lineal	10	192	0.71
Polinomial	1000	175	0.71
Radial	1	176	0.73
Sigmoidal	10	198	0.57

Nota: Solo el kernel radial con un costo de 1 genera un buen modelo. Elaborado por: (El autor, 2024).

Figura 34

Precisión vs Costo - Kernel: Radial - Costo: 1 - MSV



Nota: El Accuracy máximo es de 0.732. Elaborado por: (El autor, 2024).

Al entrenar el modelo se lo evaluó con la propia partición de entrenamiento correspondiente al 70 % de los datos. Ahora el modelo a pesar de que fue entrenado con 17 datos para la etapa 1 y 30 datos para la etapa 2, no reconoce nignuna de estas instancias, esto se puede dar debido a que esa única instancia de la creatinina sérica no es suficiente para poder abordar todas las etapas de la ERC y de hecho la propia distribución de los datos en esta variable esta afectando a este modelo.

Figura 35

Matriz de confusión Cuarto Modelo MSV

(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion entrenamiento: 206 clasificaciones correctas - Matriz de confusión prueba: 99 clasificaciones correctas. Elaborado por: (El autor, 2024).

En la figura 35 (b) se evaluó el modelo con el 30 % de los datos de prueba, en la tabla 22 se encuentra la matriz de observación y en la tabla 23 las métricas de evaluación del modelo. Por ejemplo ahora vemos que se equivoca en 21 instancias de 120, por lo cual esta clasificando bien un total de 99 instancias, es por esto que tiene un accuracy de todo el modelo del 0.82 %.

En el caso de la precisión de igual forma la etapa 1 y 2 tienen un valor del 0 %, y la etapa que mejor rendimiento presenta es la de la etapa 5 dado que tiene un valor del 100 %, esto se debe a que tiene escasos datos, de ahí la etapa 0 y 4 presentan un rendimiento por encima del 80 %, por último la etapa 3 tiene un rendimiento que se encuentro por debajo del 80 %, con un valor de 77 %, el cual a pesar de que clasifica bien las 45 instancias correspondientes a esa etapa, pero tiene un total de 9 falsos positivos que pertenecen tanto para la etapa 1, 2 y 3.

Respecto a la sensibilidad en la etapa 0 se tiene un valor de 100 % esto se debe a que en esta etapa no se tiene ningnún falso negativo, la otra etapa que también tiene pocos falsos negativos es la 3 en esta solo tiene dos falsos negativos. De ahí la etapa 4 y 5 si tienen un valor de sensibilidad un poco más bajo dado que se encuetra por debajo del 85 %, por último en el caso de la etapa 1 y 2 la sensibilidad tiene un valor del 0 % dado que no tiene verdaderos positivos.

En este modelo final, observamos que la especificidad para las etapas 1, 2 y 5 alcanza un valor del 100 %, ya que ninguna de ellas presenta falsos positivos. En contraste, las etapas 0, 3

y 4 exhiben una especificidad superior al 85 %, explicada por la presencia de 9, 10 y 2 falsos positivos, respectivamente.

Tabla 22

Matriz de observación - Cuarto modelo.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	45	66	9	0
Etapa 1	0	115	0	5
Etapa 2	0	111	0	9
Etapa 3	33	75	10	2
Etapa 4	16	99	2	3
Etapa 5	5	113	0	2

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 23

Métrica de evaluación - Cuarto modelo.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	0.86	0.92	0.00	0.00	0.00	0.00
Precisión	0.73	0.83	0.00	0.00	0.00	0.00
Medida F	0.84	0.91	0.00	0.00	0.00	0.00
Sensibilidad	1.00	1.00	0.00	0.00	0.00	0.00
Especificidad	0.78	0.88	1.00	1.00	1.00	1.00

	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	0.88	0.9	0.95	0.96	0.94	0.98
Precisión	0.70	0.77	0.83	0.89	0.81	1.00
Medida F	0.79	0.85	0.79	0.86	0.62	0.83
Sensibilidad	0.91	0.94	0.76	0.84	0.5	0.71
Especificidad	0.87	0.88	0.98	0.98	0.99	1.00
Accuracy General					0.73	0.82
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

6.4. Modelos de Redes Neuronales (ANN)

6.4.1. Modelo ANN con todas las variables

De igual forma para los modelos de ANN se consideró usar las mismas variantes que se generó en la selección de características, pero en este caso de igual forma se realizó un modelo previo con el uso de todas las características. Aplicando el método de k-fold cross validation y usando diferentes capas se obtuvo los resultados de la tabla 24. En la cual se encontro como

mejor modelo aquel que generaba un accuracy del 68 %. Para llegar a ese nivel, se tuvo que usar un total de 3 capas ocultas dado que al usar menos, el rendimiento del modelo en el k-fold cross validation se encontraba por debajo del 68 %. De igual forma algo que se considera en el modelo de ANN es el número de neuronas tanto para la capa de entrada, oculta y salida, pero el único parámetro que podemos manipular es el número de neuronas que tendra cada capa oculta.

En este modelo la estructura de la ANN es bastante compleja a tal medida que no la podemos interpretar en un gráfico, esto se debe a la gran cantidad de variables que se uso para el entrenamiento de la ANN, sobre todo en el caso de las variables de tipo cualitativas nominales, debido a que se deben de crear variables ficiticias para poder añadirlas al modelo. Por lo que en la capa de entrada, se tuvo un total de 51 neuronas, las cuales representan a las variables, en primera, segunda y tercera capa oculta se tuvieron 10, 20 y 20 neuronas respectivamente, y por último a la capa de salida se tenian un total de 6 neuronas, las cuales representan las clases a la cual al que realizar la clasificación.

Tabla 24

Pasos - Primer modelo.

Función de activación	Capas Ocultas	NC1	NC2	NC3	Mejor Accuracy
Logística	1	5	0	0	No converge
		10	0	0	0.60
		20	0	0	0.56
	2	10	10	0	0.62
		10	20	0	0.65
		10	30	0	0.63
	3	10	20	10	0.67
		10	20	20	0.68
		10	20	30	0.63

Nota: Elaborado por: (El autor, 2024).

Posterior a encontrar los hiperparámetros se considero realizar una validación adicional, aplicando el método de random cross validation, donde se tomo de los datos de entrenamiento

el 70 % para entrenar y el otro 30 % para validar. De esta forma se determinó que con 4 repeticiones o épocas en la ANN ya llegaba a un nivel estable de accuracy y en el caso de aumentar las mismas este empezaba a disminuir significativamente, por lo que el modelo se comenzaba a sobreajustar.

Tabla 25

Random Cross Validation - Primer modelo ANN.

Época	Accuracy Valdidación
1	0.6904762
2	0.6071429
3	0.6428571
4	0.7380952
5	0.702381
6	0.6547619
7	0.5833333
8	0.6309524
9	0.6309524

Nota: Elaborado por: (El autor, 2024).

Una vez entrenado el modelo se lo evaluó en la partición de entrenamiento. A diferencia del primero modelo de MSV en este caso la ANN no se equivoca en ninguna instancia, por lo cual realiza 280 clasificaciones correctamente. Es decir el modelo aprendió muy bien los datos en cada fase de entrenamiento.

Figura 36

Matriz de confusión Primer Modelo ANN

(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion prueba: 280 clasificaciones correctas - Matriz de confusión entrenamiento: 87 clasificaciones correctas. Elaborado por: (El autor, 2024).

Al evaluar el modelo con los datos de prueba se obtuvo la matriz de confusión de la figura 36 (b), de igual forma se obtuvo la matriz de observación en la tabla 26 y por último se obtuvo las métricas de evaluación en la tabla 27. En este caso el modelo se equivoca en 33 instancias de 120, lo cual quiere decir que a 87 instancias si las clasificó bien. En el tema del accuracy vemos que tiene un rendimiento general del 72 %.

Al hablar de la precisión, la única que tiene un buen rendimiento es la de la etapa 0, ya que tiene un valor del 98 %, de igual forma este valor alto se debe a que solo tiene 1 falso positivo dentro de sus clasifiaciones, es decir de las 45 instancias que el modelo dijo que son de la etapa 0, en realidad 1 no pertenecia a esa etapa. En las etapas 3 y 4 estas almenos superaban el 70 %, ya que tenian un rendimiennto del 76 % y 77 % respectivamente. En el caso de la etapa 1, 2 y 5, el rendimiento de estas se encontraba por debajo del 60 %, es decir en realidad el modelo no produce clasificaciones consistentes en estas etapas.

La sensibilidad en el modelo es bastante baja para las etapas que van desde la 2 hasta la 5, esto es causado a la cantidad de falsos negativos que poseen, por lo cual en estas etapas se esta equivocando en la clasificación en un gran cantidad de individuos respecto a sus etapas. La única etapa que si tiene una buena sensibilidad es la 0, ya que tienen un valor del 98 %, esto gracias a que solo tiene 1 falso negativo en su clasificación.

Por último en el caso de la especificidad, la mayoría de los modelos tienen un valor

por encima del 85 %, dado a que tienen una mayor cantidad de verdaderos negativas en comparación a un menor número de falsos positivos por cada etapa.

Tabla 26

Matriz de observación - Primer modelo ANN.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	44	74	1	1
Etapa 1	4	112	3	1
Etapa 2	3	98	13	6
Etapa 3	22	78	7	13
Etapa 4	10	98	3	9
Etapa 5	4	107	6	3

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 27

Métrica de evaluación - Primer modelo ANN.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	1.00	0.98	1.00	0.97	1.00	0.84
Precisión	1.00	0.98	1.00	0.57	1.00	0.19
Medida F	1.00	0.98	1.00	0.67	1.00	0.24
Sensibilidad	1.00	0.98	1.00	0.80	1.00	0.33
Especificidad	1.00	0.99	1.00	0.97	1.00	0.88
	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	1.00	0.83	1.00	0.90	1.00	0.92
Precisión	1.00	0.76	1.00	0.77	1.00	0.40
Medida F	1.00	0.69	1.00	0.62	1.00	0.47
Sensibilidad	1.00	0.63	1.00	0.53	1.00	0.57
Especificidad	1.00	0.92	1.00	0.97	1.00	0.95
Accuracy General					1.00	0.72
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

6.4.2. Modelo ANN con primer variante

Como segundo modelo al verificar que hiperparámetros genereban un buen rendimiento con tan solo usar 6 variables se obtuvo la tabla 28, en la cual al aplicar el método de k-fold cross validatión se vio que había un mejor rendimiento usando una función de activación logísica, en cuanto el número de capas ocultas pues al intentar de aumentar más de una capa oculta los modelos se hacian tan complejos a tal punto que no lograban converger.

Tabla 28

Pasos - Segundo modelo.

Función de activación	Capas Ocultas	NC1	NC2	NC3	Mejor Accuracy
Logística	1	5	0	0	0.75
		10	0	0	No converge
	2	5	5	0	No converge
Tangente Hiperbólica	1	5	0	0	0.74
		5	5	0	No converge

Nota: Elaborado por: (El autor, 2024).

Aplicando una validación adicional para determinar el número de épocas necesarios a realizar antes de llegar a un sobreajuste, se obtuvó un accuracy del 69 % con tan solo una iteración. Y en el caso de aumentar las épocas, este valor se mantenía y disminuía, de forma que no es necesario aumentar las épocas y generar un sobreajuste en el modelo.

Tabla 29

Random Cross Validation - Segundo modelo ANN.

Época	Accuracy Valdidación
1	0.6904762
2	0.6785714
3	0.6904762
4	0.6309524
5	0.6309524
6	0.5952381

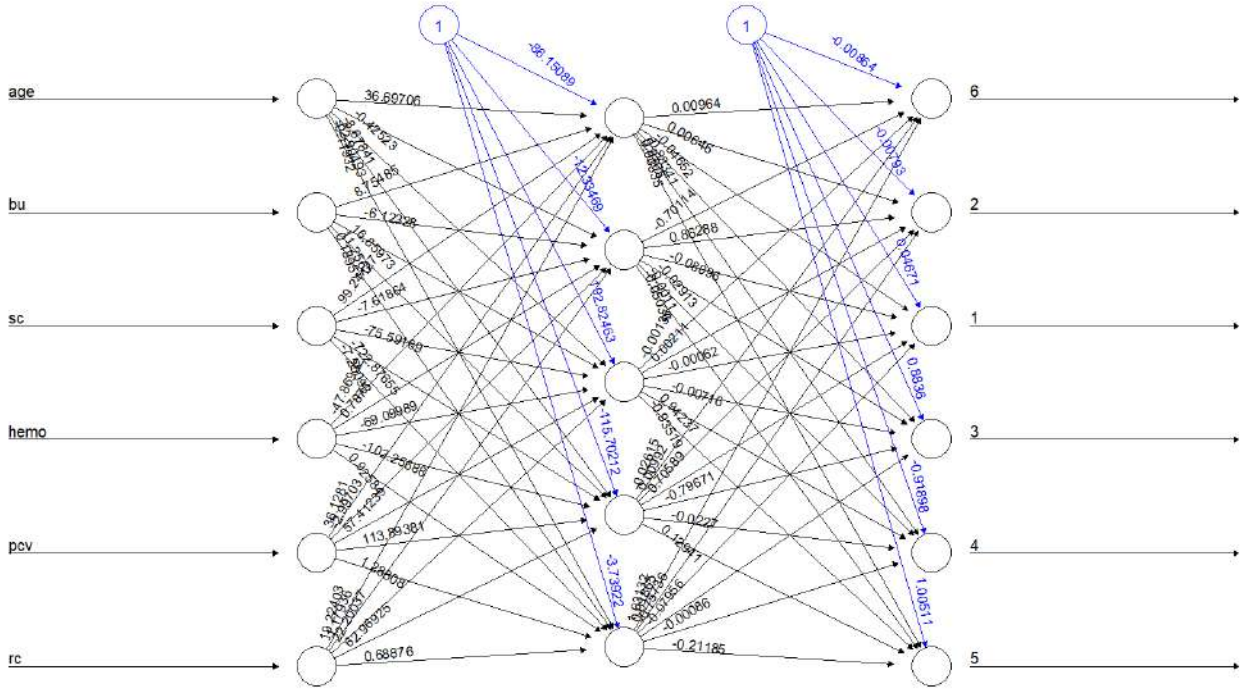
Nota: Elaborado por: (El autor, 2024).

En relación con la estructura de la ANN en el segundo modelo, se ilustra en la figura 37. Como se puede apreciar, la capa de entrada está compuesta por 6 neuronas, que representan las variables empleadas para entrenar el modelo. En cuanto a la capa oculta, se optó por una

única capa que contiene un total de 5 neuronas. Finalmente, en la capa de salida, se tienen 6 neuronas que representan las distintas etapas que el modelo debe clasificar.

Figura 37

Estructura ANN - Segundo Modelo



Nota: Capa de entrada: 6 neuronas - capa oculta: 5 neuronas - capa de salida: 6 neuronas. Elaborado por: (El autor, 2024).

Con el modelo ya entrenado, se procedió a evaluarlo con la partición de entrenamiento, obteniendo como resultado la matriz de confusión de la figura 38 (a). En este caso, el modelo se equivoca en un total de 37 instancias de 280, logrando clasificar correctamente 243 instancias en sus correspondientes etapas. En la etapa 5, acierta al clasificar 13 instancias de las 26; es decir, el resto fue clasificado erróneamente en otras etapas. Este tipo de errores es muy grave, ya que el modelo indica que personas que necesitarían un proceso de diálisis en realidad no lo necesitan, al considerar que están por debajo de la etapa 5.

Figura 38

Matriz de confusión Segundo Modelo ANN

(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion prueba: 243 clasificaciones correctas - Matriz de confusión entrenamiento: 102 clasificaciones correctas. Elaborado por: (El autor, 2024).

Evaluando el modelo con la partición de prueba se obtuvo en la figura 38 (b) la matriz de confusión, en la tabla 30 la matriz de observación y en la tabla 31 las métricas de evaluación. El modelo cometió 18 errores de un total de 120 instancias, logrando un accuracy general del 85 %.

En este caso la precisión es buena tanto para la etapa 0, 3, 4 y 5, dado que estas tiene un rendimineto entre el 80 % y el 100 %, siendo la etapa 5 la que tiene el 100 % del rendimiento, esto se debe a que el modelo no tienen ningun falso postivo y a demás a que logra clasificar bien a 4 instancias. En el restante de las etapas la que peor rendimiento presenta es la etapa 1 ya que tiene un valor del 50 % de precisión, esto se debe a que ese 50 % de los datos que esta clasificando mi modelo no corresponde a esa etapa, por lo cual en este caso, esta es etapa que más dificultades tiene el modelo para clasificar.

Las únicas etapas que cumplen un buen nivel de sensibilidad son la 0, 2, 3 y 4 dado que tienen un valor del 93 %, 89 %, 80 % y del 95 % respectivamente. En cambio en la etapa 1 se tiene un valor del 40 %, dado que tiene un total de 3 falsos negativos con respecto a solo 2 verdaderos positivos. En el caso de la etapa 5 se tiene un valor del 57 %, dado que tiene 3 falsos negativos con respecto a 4 verdaderos positivos.

En la especificidad todas las etapas superán el 90 % del rendimiento e inclusive la etapa 5

es la única que llega al 100 %. Con estas métricas podemos decir que a medida general las etapas tienden a evitar los falsos positivos de forma efectiva.

Tabla 30

Matriz de observación - Segundo modelo ANN.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	42	71	4	3
Etapa 1	2	113	2	3
Etapa 2	8	106	5	1
Etapa 3	28	82	3	7
Etapa 4	18	97	4	1
Etapa 5	4	113	0	3

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 31

Métrica de evaluación - Segundo modelo ANN.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	0.95	0.94	0.97	0.96	0.92	0.95
Precisión	0.90	0.91	0.78	0.50	0.64	0.61
Medida F	0.94	0.92	0.71	0.44	0.62	0.73
Sensibilidad	0.99	0.93	0.65	0.40	0.60	0.89
Especificidad	0.93	0.95	0.98	0.98	0.96	0.95
	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	0.95	0.92	0.98	0.96	0.95	0.97
Precisión	0.87	0.90	0.94	0.82	1.00	1.00
Medida F	0.91	0.85	0.94	0.88	0.67	0.73
Sensibilidad	0.96	0.80	0.94	0.95	0.50	0.57
Especificidad	0.95	0.96	0.99	0.96	1.00	1.00
Accuracy General					0.87	0.85
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

6.4.3. Modelo ANN con segunda variante

Al aplicar el método de validación en el tercer modelo de ANN se obtuvo la tabla 32, con el uso de 2 variables, el comportamiento de los hiperparámetros fue el mismo que en el anterior modelo con 6 varibles, con la única diferencia que ahora la función de activación que tenía una mejor respuesta era la de la tangente hiperbólica. Es por esto que se tomó en cuenta como mejores hiperparámetros a los de la tabla 32, donde se tenía a una capa oculta

con 5 neuronas que generaban un rendimiento del accuracy del 76 %.

Tabla 32

Pasos - Tercer modelo.

Función de activación	Capas Ocultas	NC1	NC2	NC3	Mejor Accuracy
Logística	1	5	0	0	0.76
		10	0	0	No converge
	2	5	5	0	No converge
Tangente Hiperbólica	1	5	0	0	0.77
	2	5	5	0	No converge

Nota: Elaborado por: (El autor, 2024).

En este caso después de la validación adicional se llegó hasta 2 épocas, de forma que el rendimiento cambiaba en la segunda época a un valor del 77 % y se mantenía en el caso de que las épocas aumenten.

Tabla 33

Random Cross Validation - Tercer modelo ANN.

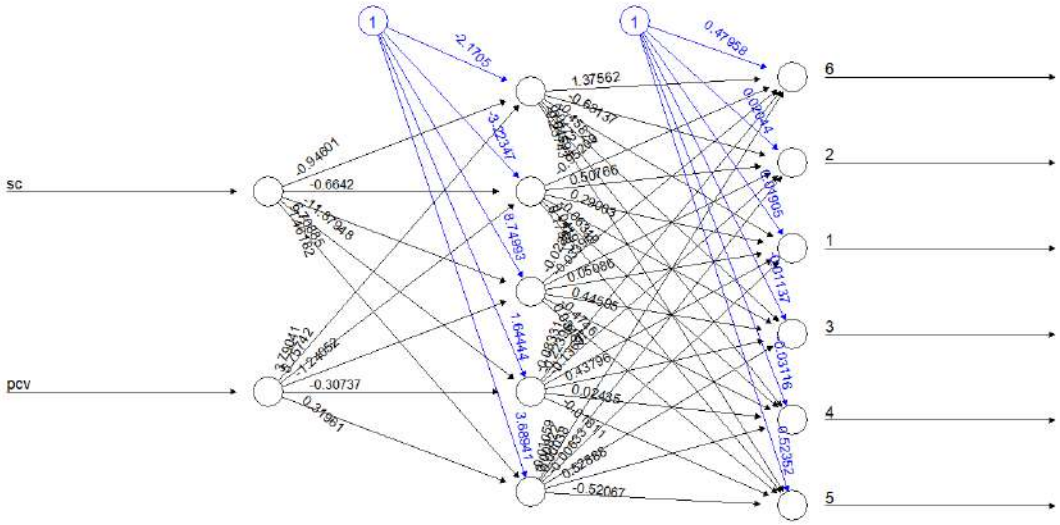
Época	Accuracy Valdidación
1	0.7261905
2	0.7738095
3	0.7738095
4	0.7738095
5	0.7261905
6	0.7619048
7	0.7619048

Nota: Elaborado por: (El autor, 2024).

En este modelo la estructura de la ANN se simplificó aún más. En la capa de entrada, se redujo a solo dos neuronas, representando las dos variables utilizadas para el entrenamiento del modelo. La capa oculta se mantiene con 5 neuronas, y en la capa de salida, continúan las 6 neuronas para clasificar las diferentes etapas.

Figura 39

Estructura ANN - Tercer Modelo



Nota: Capa de entrada: 2 neuronas - capa oculta: 5 neuronas - capa de salida: 6 neuronas. Elaborado por: (El autor, 2024).

Una vez entrenado el modelo se lo evaluó en la partición de entrenamiento, teniendo como resultado la matriz de confusión de la figura 40 (a). En este caso el modelo se equivoca en 60 instancias de 280. Algunas observaciones preocupantes en esta matriz es el hecho de que tiene 14 instancias que pertenecen a la etapa 5 pero el modelo las clasifica en etapas como la 2,3 y 4, las dos primeras son muy distantes a la 5 y además estas instancias en realidad son personas de alto riesgo, pero el modelo no los considera como tal.

Figura 40

Matriz de confusión Tercer Modelo ANN

(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion prueba: 220 clasificaciones correctas - Matriz de confusión entrenamiento: 98 clasificaciones correctas. Elaborado por: (El autor, 2024).

La evaluación del modelo con la partición de prueba se presenta en la figura 40 (b) con la matriz de confusión, en la tabla 34 con la matriz de observación y en la tabla 35 con las métricas de evaluación. Este modelo cometió alrededor de 22 clasificaciones incorrectas de 120 instancias, logrando un rendimiento general de precisión del 82 %.

En la precisión las únicas etapas que si tienen un buen nivel de precisión son la etapa 0, 3, 4 y 5, dado que superan el 70 % de rendimiento. En las otras etapas la que más baja precisión es la de la 2, ya que tiene un valor del 0.33, esto se da por que tiene tan solo 1 verdadero positivo con respecto a los 2 falsos positivos que tuvo en su clasificación. Algo similiar sucede con la etapa 1 dado que tiene un valor del 50 %, y esto se da, ya que el número de verdaderos positivos con el de falsos positivos es el mismo, en este caso un valor de 2.

Las etapas que muestran un nivel de sensibilidad adecuado son la 0, 3 y 4, ya que su rendimiento supera el 80 %. Por otro lado, las etapas 1, 2 y 5 presentan un rendimiento inferior al 60 %. Destaca la etapa 2 como la de peor rendimiento, ya que se observa un alto número de falsos negativos, con un total de 8 instancias que debían clasificarse en la etapa 2, pero el modelo cometió errores. Se observa una situación similar en las etapas 1 y 5, aunque en menor medida. Estos resultados indican que el modelo no es completamente fiable en estas etapas.

En cuanto a la especificidad, la mayoría de las etapas superan el 90 %, destacando que la etapa 5 alcanza un 100 % en esta métrica. Además, la etapa 3 logra un valor del 88 % en este

aspecto. En términos generales, estas métricas indican que el modelo tiende a evitar los falsos positivos de manera efectiva.

Tabla 34

Matriz de observación - Tercer modelo ANN.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	44	70	5	1
Etapa 1	2	113	2	3
Etapa 2	1	109	2	8
Etapa 3	31	75	10	4
Etapa 4	16	98	3	3
Etapa 5	4	113	0	3

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 35

Métrica de evaluación - Tercer modelo ANN.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	0.94	0.95	0.95	0.96	0.88	0.92
Precisión	0.87	0.90	0.61	0.50	0.40	0.33
Medida F	0.93	0.94	0.63	0.44	0.20	0.17
Sensibilidad	0.99	0.98	0.65	0.40	0.13	0.11
Especificidad	0.91	0.93	0.97	0.98	0.98	0.98

	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	0.88	0.88	0.96	0.95	0.94	0.97
Precisión	0.71	0.76	0.87	0.84	0.86	1.00
Medida F	0.80	0.81	0.82	0.84	0.60	0.73
Sensibilidad	0.91	0.88	0.79	0.84	0.46	0.57
Especificidad	0.88	0.88	0.98	0.97	0.99	1.00
Accuracy General					0.78	0.82
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

6.4.4. Modelo ANN con tercera variante

Como último modelo se considero tan solo usar la característica de la creatinina sérica, para ver cual era el rendimiento del modelo en este caso. Al aplicar el método de validación cruzada k-fold, se determinó que solo una función de activación permitía la convergencia del modelo. En cuanto a las capas, se observó que solo una capa oculta con 5 neuronas era necesaria para lograr un accuracy del 72 %. De hecho, al intentar aumentar la complejidad del

modelo mediante la adición de más capas ocultas con más neuronas, se encontró dificultad en lograr la convergencia del modelo. Es por esto que los mejores hiperparámetros para el modelado de la ANN son los de la tabla 37.

Tabla 36

Pasos - Cuarto modelo.

Función de activación	Capas Ocultas	NC1	NC2	NC3	Mejor Accuracy
Logística	1	5	0	0	0.72
		10	0	0	No converge
	2	5	5	0	No converge
Tangente Hiperbólica	1	5	0	0	No converge
	2	5	5	0	No converge

Nota: Elaborado por: (El autor, 2024).

En este caso con solo una época ya se tenia el mejor accuracy en la parte de validación, con un valor del 72 %. Después de 5 iteraciones este comportamiento se mantuvo igual así que solo se considero una época para evitar el sobreajuste del modelo.

Tabla 37

Random Cross Validation - Cuarto modelo ANN.

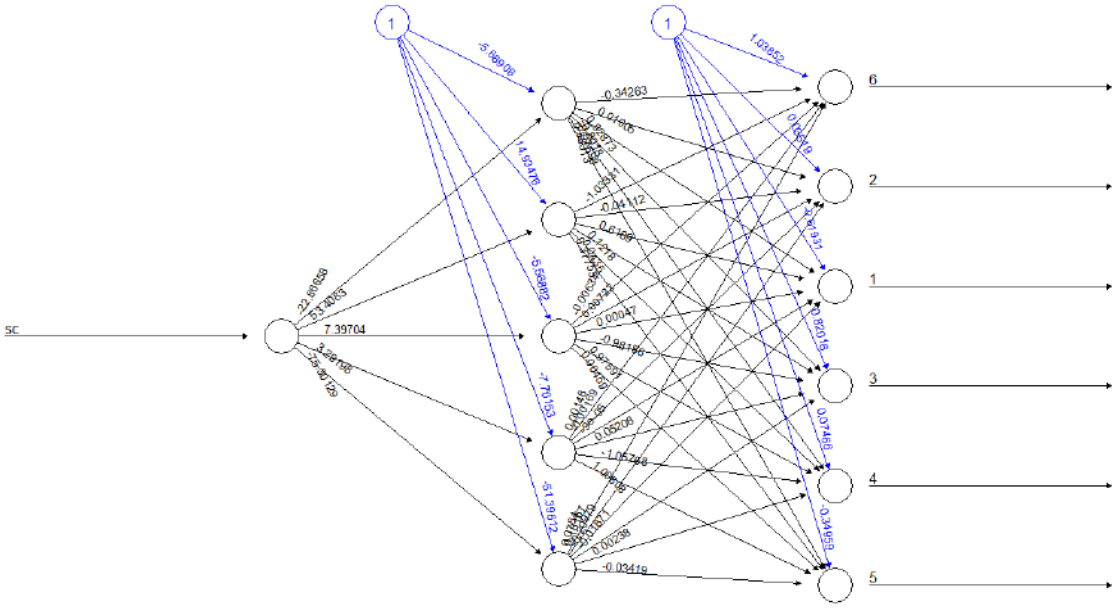
Época	Accuracy Valdidación
1	0.7261905
2	0.7261905
3	0.7261905
4	0.7261905
5	0.7261905
6	0.7261905

Nota: Elaborado por: (El autor, 2024).

En cuanto a la estructura y la complejidad de esta ANN se encuentra en la figura 41, como vemos ahora es más simple dado que en la capa de entrada tiene tan solo una neurona, en la capa oculta continua con 5 neuronas y como siempre en la capa de salida tiene las 6 neuronas donde se realizará la clasificación.

Figura 41

Estructura ANN - Cuarto Modelo



Nota: Capa de entrada: 1 neuronas - capa oculta: 5 neuronas - capa de salida: 6 neuronas. Elaborado por: (El autor, 2024).

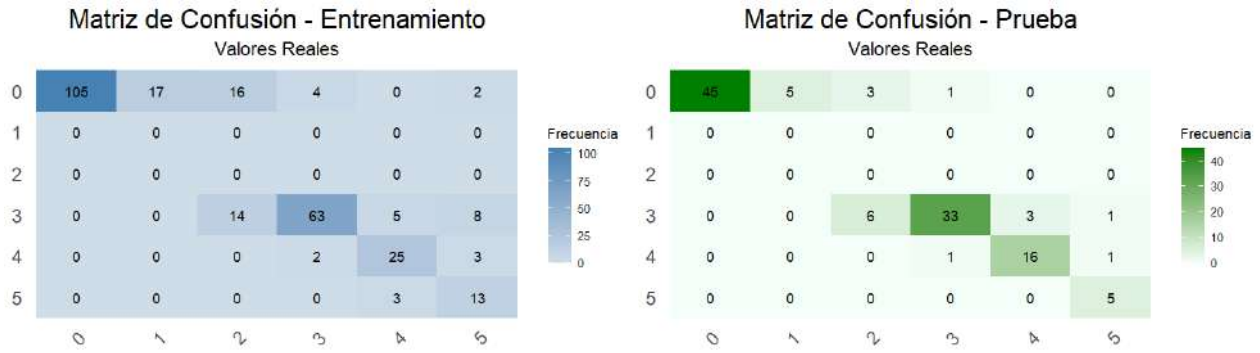
Evaluando al modelo con los mismos datos con los que se entrenó, se evidenció que la matriz de confusión de la figura 42 (a) generaba las mismas clasificaciones de la matriz de confusión de la figura 35, es decir los resultados del rendimiento del último modelo de MSV donde se uso una sola variable que es la creatinina sérica. Este fenómeno puede explicarse por la naturaleza linealmente separable del conjunto de datos y la simplicidad que aporta la utilización exclusiva de esta única característica. En consecuencia, tanto SVM como ANN logran ajustarse de manera similar al conjunto de datos.

Una vez más, se observa que en las etapas 1 y 2, el modelo no logra clasificar ninguna

instancia. En la etapa 0, se identifican algunos falsos positivos, lo cual es problemático ya que excluye a 39 personas de un diagnóstico de Enfermedad Renal Crónica (ERC) durante la propia fase de entrenamiento. En cuanto a la etapa 5, se registra la clasificación errónea de 3 personas, asignándolas a esta etapa en lugar de la 4. En términos generales, el modelo muestra dificultades para aprender las instancias de entrenamiento correspondientes.

Figura 42

Matriz de confusión Cuarto Modelo ANN



(a) *Entrenamiento* **(b)** *Prueba*

Nota: Matriz de confusion prueba: 206 clasificaciones correctas - Matriz de confusión entrenamiento: 99 clasificaciones correctas. Elaborado por: (El autor, 2024).

En la figura 42 (b), se llevó a cabo la evaluación del modelo utilizando el 30 % de los datos de prueba. La matriz de observación se presenta en la tabla 38, mientras que las métricas de evaluación del modelo se detallan en la tabla 39. Es relevante destacar que de igual forma en este caso, la matriz de confusión obtenida al evaluar el modelo con el conjunto de datos de prueba es idéntica a la matriz de confusión del último modelo de SVM. Por lo tanto, los valores tanto del accuracy general como de los individuales se mantendrán, es decir, un valor general del 82 %. De igual forma lo mismo sucede en el caso de la precisión, senbilidad y la especificidad, estos valores se mantendrán iguales al del último modelo de SVM.

Tabla 38

Matriz de observación - Cuarto modelo ANN.

	Medidas			
Etapas	TP	TN	FP	FN
Etapa 0	45	66	9	0
Etapa 1	0	115	0	5
Etapa 2	0	111	0	9
Etapa 3	33	75	10	2
Etapa 4	16	99	2	3
Etapa 5	5	113	0	2

Nota: TP: Verdadero Positivo, TN: Verdadero Negativo, FP: Falso Positivo, FN: Falso Negativo. Elaborado por: (El autor, 2024).

Tabla 39

Métrica de evaluación - Cuarto modelo ANN.

	Etapa 0		Etapa 1		Etapa 2	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	105	45	17	5	30	9
Accuracy	0.86	0.92	0.00	0.00	0.00	0.00
Precisión	0.73	0.83	0.00	0.00	0.00	0.00
Medida F	0.84	0.91	0.00	0.00	0.00	0.00
Sensibilidad	1.00	1.00	0.00	0.00	0.00	0.00
Especificidad	0.78	0.88	1.00	1.00	1.00	1.00

	Etapa 3		Etapa 4		Etapa 5	
	Entrenamiento	Prueba	Entrenamiento	Prueba	Entrenamiento	Prueba
# Instancias	69	35	33	19	26	7
Accuracy	0.88	0.9	0.95	0.96	0.94	0.98
Precisión	0.70	0.77	0.83	0.89	0.81	1.00
Medida F	0.79	0.85	0.79	0.86	0.62	0.83
Sensibilidad	0.91	0.94	0.76	0.84	0.5	0.71
Especificidad	0.87	0.88	0.98	0.98	0.99	1.00
Accuracy General					0.73	0.82
Total Instancias					280	120

Nota: Evaluación del modelo con el conjunto de entrenamiento y prueba. Elaborado por: (El autor, 2024).

7. RESULTADOS Y DISCUSIONES

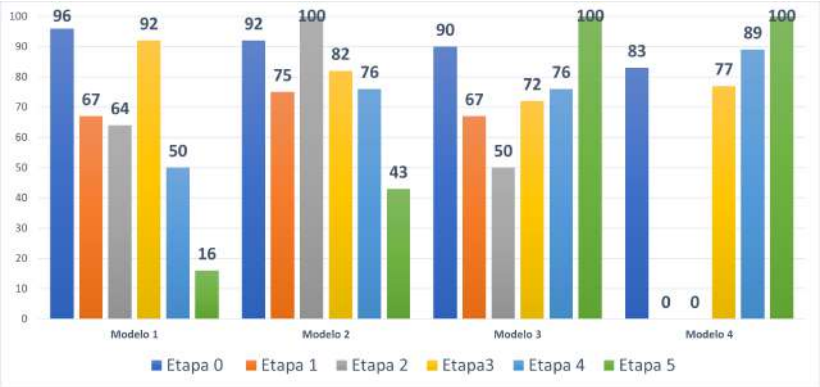
7.1. Rendimiento de modelos de máquinas de soporte vectorial

El modelo que mejor rendimiento tiene en cuestión de accuracy para la mayoría de etapas es la del modelo 2 dado que tiene un mejor rendimiento en todas sus etapas, especificamente mayor al 90 %, pero con solo ver el accuracy de una clase no podemos definir que un modelo es

bueno o no, dado que inclusive el accuracy con set de datos desbalanceados puede enmascarar el comportamiento del desempeño en el modelo para diferentes categorías, es por esto que se le dará más importancia a las demás métricas.

Figura 43

Comparación de precisión en modelos de SVM



Nota: Precisión en % para los cuatro modelos de SVM. Elaborado por: (El autor, 2024).

La precisión si que es una métrica de importancia sobre todo por el desbalance que se tiene entre las etapas. Como se observa en la figura 43 en el primer modelo las únicas etapas que cumplen con un buen rendimiento son la 0 y la 3 ya que se encuentran en 96 % y 92 % respectivamente. Pero etapas como la 1, 2, 4 y sobre todo la 5 tienen un mal rendimiento, por lo cual, no es un buen modelo, el hecho de usar todas las variables en este modelo de SVM generá que aparte de ser complejo, genera sobre ajuste para las etapas con un rendimiento bajo.

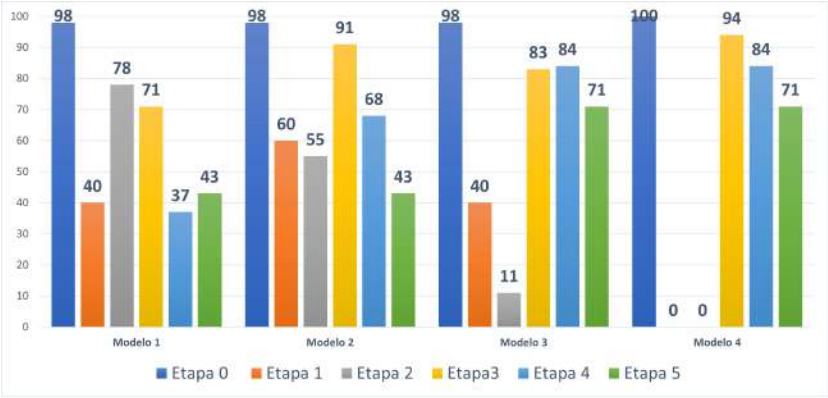
En el caso del modelo 2 las etapas desde la 0 hasta la 4 se encuentran dentro de un rendimiento entre bueno y aceptable, es decir del 70 % al 100 %, el problema es que la etapa 5 tiene un rendimineto del 43 %, el cual es muy bajo, esto quiere decir que en la etapa 5 el modelo tiende a tener una gran cantidad de falsos positivos, lo cual es malo dado que en esta etapa los pacientes a este nivel ya requieren de una intervención de diálisis, si se tomará a este modelo como una referencia para diagnóstico de personas con ERC, a todas las personas que el modelo clasifique en etapa 5 tan solo el 43 % realmente pertenecerán a esa etapa.

En cambio en el modelo 3 vemos que las únicas etapas que tienen un rendimiento bueno o aceptable son la etapa 0, 3, 4 y 5, ya que tiene un valor del 90%, 72%, 76% y 100% respectivamente. A diferencia del modelo 2 vemos que las clasificaciones que haga el modelo en la etapa 5, ahora el 100% de los datos,si corresponderán a esa etapa. Ahora se ve una gran reducción en el rendimiento en las etapas 1 y 2, pero aun así el hecho de tener una precisión del 100% en la etapa 5 ya es un punto a favor para este modelo.

Por último en el modelo 4 vemos que en realidad la única característica que era la creatinina sérica, no fue suficiente para poder generar un buen modelo, dado que para la etapa 1 y 2 tienen un rendimiento del 0%, para las demas etapas si tienen un buen rendimiento, pero por la falta de precisión en la etapa 1 y 2, a este modelo simplemente no se lo podría considerar como una opción.

Figura 44

Comparación de sensibilidad en modelos de SVM



Nota: Sensibilidad en% para los cuatro modelos de SVM. Elaborado por: (El autor, 2024).

Al igual que la precisión, la sensibilidad también es una métrica de importancia al momento de evaluar modelos con datos desbalanceados. El modelo 1 tiene 3 etapas con un rendimiento aceptable el cual es la etapa 0, 2 y 3, dado que se encuentran entre el 70% y el 100%. Pero en el caso de la etapa 1, 4 y 5 su sensibilidad se encuentra por debajo del 50%, es decir en general este es un mal modelo ya que cometara una gran cantidad de falsos negativos sobre todo en las etapas con más bajo rendimiento.

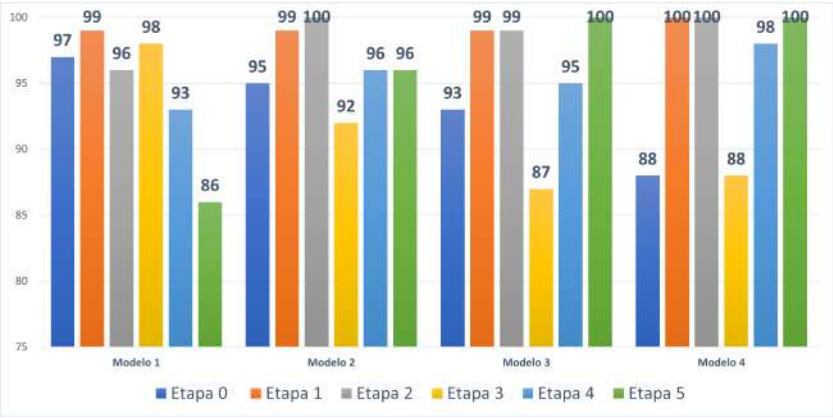
En el caso del modelo 2 tan solo tiene dos etapas con un buen rendimiento, la etapa 0 y la etapa 3, con valores del 98 % y 91 % respectivamente. Pero en las demas etapas su rendimiento se encuentra por debajo del 70 %, sobre todo en la etapa 5 que es una de gran importancia tiene un valor del 43 %, es decir que las personas que realmente tienen esta etapa, tan solo el 43 % son clasificadas como si la tuvieran.

En el modelo 3 nuevamente se tiene un buen rendimiento en las etapas 0, 3, 4 y a estas se suma también la etapa 5, con un valor del 98 %, 83 %, 84 % y 71 % respectivamente. El rendimiento para la etapa 1 con respecto a los anteriores modelos se mantuvo, pero el de la etapa 3, 4 y 5 sí que tuvo un aumento significativo, por ejemplo en la etapa 5 tuvo un aumento con respecto al modelo 2 con 28 puntos porcentuales. Es por esto que el modelo 3 por ahora sigue siendo el mejor modelo.

Por último algo que llama la atención en el modelo 4 es que tiene una sensibilidad del 100 % para la etapa 0, es decir que el 100 % de los casos que realmente son positivos, el 100 % de ellos los clasificará como positivos. Y de igual forma el rendimiento en la etapa 1 y 2 seguirán siendo 0 %, por esta razón este es un modelo que no se lo puede considerar como aceptable.

Figura 45

Comparación de especificidad en modelos de SVM

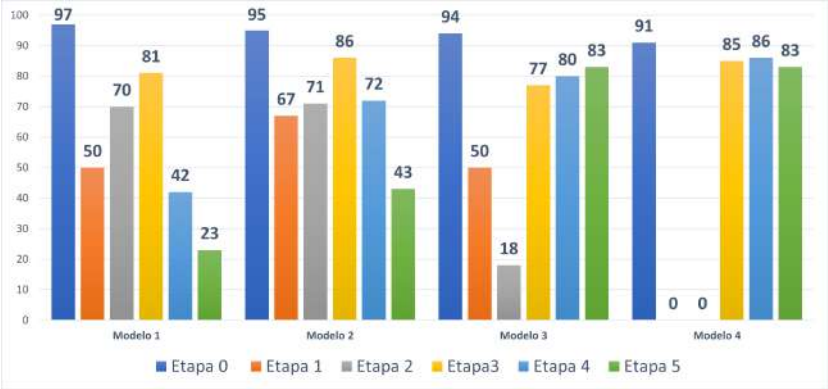


Nota: Especificidad en % para los cuatro modelos de SVM. Elaborado por: (El autor, 2024).

La especificidad al ser una métrica que nos dice que tan bueno es el modelo para identificar correctamente a los casos negativos, vemos que la mayoría de todos los modelos tiene un valor por encima del 80 %, por lo cual de todos los modelos el cual se considera con un buen valor de sensibilidad es el del modelo 3, dado que para la etapa 5 tienen una especificidad del 100 %, es decir que en la clasificación de la etapa 5 el 100 % de las instancias clasificadas corresponderan a la misma, es decir no tiene ningun falso positivo, y en las demás etapas de igual forma tienen un buen rendimiento.

Figura 46

Comparación de Medida F1 en modelos de SVM

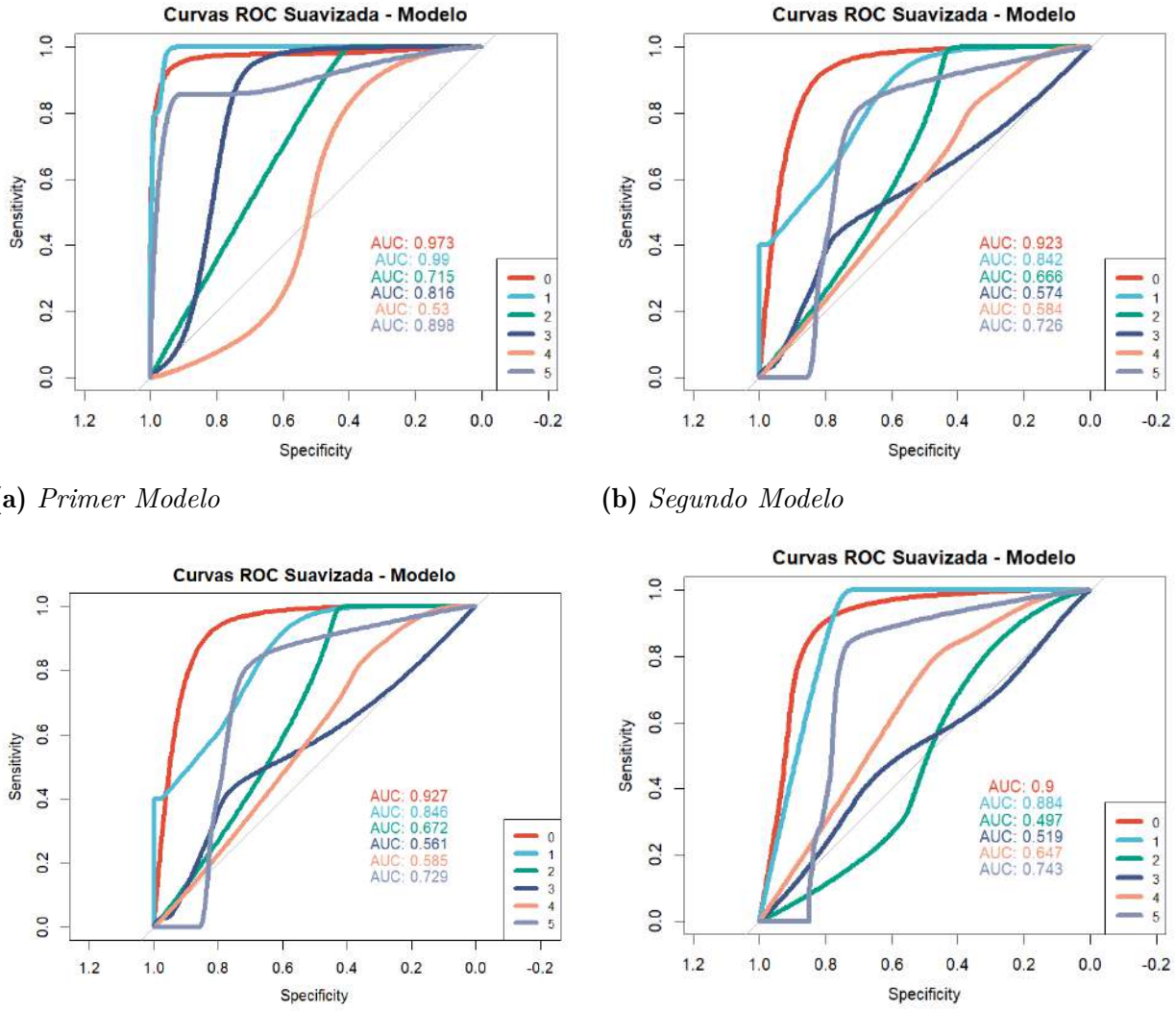


Nota: Medida F1 en % para los cuatro modelos de SVM. Elaborado por: (El autor, 2024).

El uso de la métrica de F1-score se realiza para seleccionar el modelo que tenga la menor cantidad de falsos positivos y falsos negativos. En la Figura 46, el modelo 4 destaca al exhibir la mejor medida de F1, especialmente en etapas cruciales como la etapa 0 y la etapa 5, donde alcanza un rendimiento del 94 % y 83 %, respectivamente. Pero en el caso de las etapas 1 y 2 tienen un valor del 0 %, por lo cual en estas habrá una gran cantidad de falsos negativos y positivos. Otro modelo en el que se tiene un buen nivel de F1 es el modelo 3 dado que para etapas como la 1, 3, 4 y 5 tienen un valor por encima del 70 %, y tan solo en dos etapas la 1 y la 2 su valor es del 50 % y 18 %, respectivamente.

Figura 47

Comparación curva ROC y valor AUC en modelos SVM



(a) *Primer Modelo*

(b) *Segundo Modelo*

(c) *Tercer Modelo*

(d) *Cuarto Modelo*

Nota: Elaborado por: (El autor, 2024).

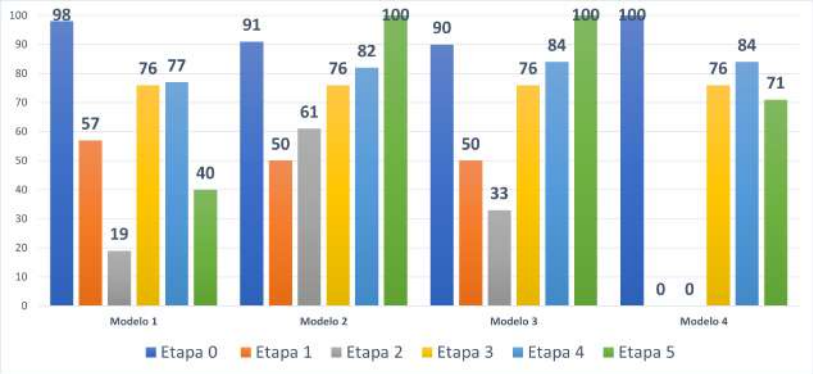
Tomando en cuenta el área bajo la curva ROC se considera como el mejor modelo, al tercer, dado que con tan solo 2 variables: creatinina sérica y volumen de células empaquetadas, esta logrando lo mismo que el segundo modelo con 6 características, las únicas diferencias son tan solo un punto porcentual en el valor AUC de la etapa 2 y 3, los demas modelos no se toman en cuenta debido a que inclusive tienen un valor por debajo del 50 %.

7.2. Rendimiento de modelos con redes neuronales

En el caso de las ANN, el modelo que generaba un buen rendimiento en términos de accuracy para todas las etapas fue la del modelo 2, dado que superan en rendimiento el 90 % del accuracy. Pero aún así esta única métrica no es suficiente para determinar el desempeño del modelo, de hecho en datos desbalanceados como en este caso es mejor ver otras métricas como la precisión y la sensibilidad.

Figura 48

Comparación de precisión en modelos de ANN



Nota: Precisión en % para los cuatro modelos de ANN. Elaborado por: (El autor, 2024).

En cuanto a la precisión en el modelo 1 se tiene como un rendimiento aceptable al de la etapa 0, 3 y 4 ya que tienen valores del 98 %, 76 % y 77 %, respectivamente. En el caso de las etapas 1, 2 y 5, el modelo tiene métricas muy bajas dado que estan por debajo del 60 %, es decir para todas esas etapas con bajo rendimiento se tendra una gran cantidad de falsos positivos, por lo que en realidad el modelo 1, no es un modelo aceptable.

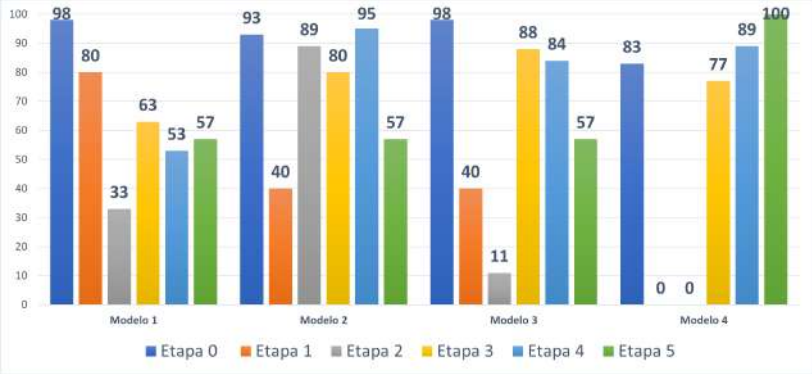
En el caso del modelo 2, se tiene buenas métricas para la etapa 0, 3, 4 y 5 con valores del 91 %, 76 %, 82 % y el 100 %, respectivamente. En el caso de las etapa 1 y 2 se tienen valores del 50 % y 61 %, respectivamente, pero aun así la etapa 2 termina siendo la que mejor rendimiento presenta a nivel de todos los modelos, de igual forma en la etapa 5 se tiene un rendimiento del 100 %, lo cual es bueno para una etapa de bastante importancia.

En cambio en el modelo 3 las únicas etapas que tienen un rendimiento bueno o aceptable son la etapa 0, 3, 4 y 5, ya que tiene un valor del 90 %, 76 %, 84 % y 100 % respectivamente. En el caso de la etapa 1 y 2, se tienen valores del 50 % y 33 %, en la etapa 2 se tiene 28 puntos porcentuales menos que en el modelo 2, por lo cual el modelo 2 por ahora sigue siendo el mejor.

Por último en el modelo 4 de igual forma que en los modelos de SVM tampoco llega a clasificar niguna instancia en la etapa 1 y 2, por lo cual su rendimiento es del 0 %. A diferencia de todos los modelos este es el único que llega a un rendimiento del 100 % en la etapa 0, pero este sigue teniendo como punto negativo el hecho de que no clasifica bien la etapa 1 y 2, por lo cual tampoco podemos considerar a este modelo como una opción a considerar.

Figura 49

Comparación de sensibilidad en modelos de ANN



Nota: Sensibilidad en % para los cuatro modelos de ANN. Elaborado por: (El autor, 2024).

La sensibilidad en el primer modelo de ANN tiene buenos valores, tan solo para la etapa 0 y 1, con un valor del 98 % y 80 %. En cambio, las demas etapas desde la 2 hasta la 5, tienen métricas por debajo del 70 %, la que peor rendimiento tiene es la etapa 2 ya que tiene un valor del 33 %, el cual es muy bajo como para dejarlo pasar, y sobre todo en la etapa 5 tiene un rendimiento del 57 %, es decir que todo lo que clasifique como etapa 5 tan solo el 57 % realmente perteneceran a esa etapa.

En el caso del modelo 2, las métricas de sensibilidad tienen una mejora significativa sobre todo para la etapa 2, 3 y 4, ya que tienen un aumento del 56 %, 17 % y del 42 %,

respectivamente. Este modelo tiene una reducción en su rendimiento del 40 % para la etapa 1 y en el caso de la etapa 5 el rendimiento se mantiene en el 57 %.

Para el modelo 3 se ve una gran reducción del 78 % de sensibilidad para la etapa 2, por lo cual ahora este modelo tendrá varias dificultades para poder clasificar bien a esta etapa. De igual forma en la etapa 4 se tiene una pequeña reducción del rendimiento en un 11 %. En el caso de la etapa 1 y 3 si que se tiene un aumento del 5 % y 8 % respectivamente.

Por último en el modelo 4 la sensibilidad para las etapas 1 y 2 es del 0 %, de ahí las demás etapas si tienen un buen rendimiento por encima del 70 %, de hecho en la etapa 5 tiene un rendimiento del 100 %, pero de todas formas como se uso una sola característica, el modelo no es bueno y confiable para las etapas 1 y 2.

Figura 50

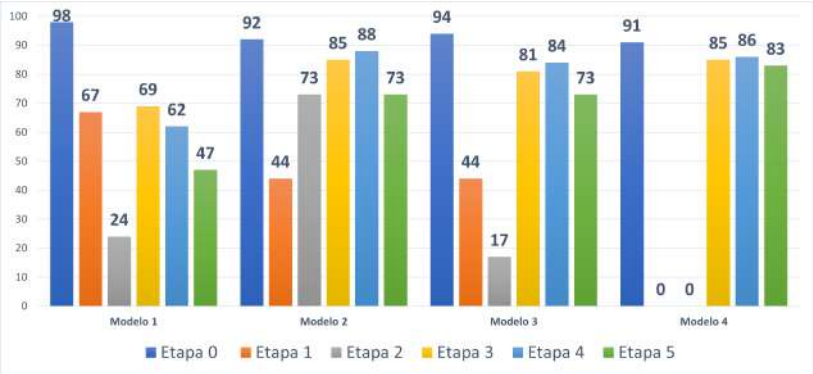
Comparación de especificidad en modelos de ANN

Nota: Especificidad en % para los cuatro modelos de ANN. Elaborado por: (El autor, 2024).

En el caso de la especificidad en todos los modelos se tiene un rendimiento por encima del 80 %, pero el modelo que en todas las etapas tiene una especificidad mayor al 90 % es en el modelo 2, sobre todo en la etapa 5 tiene un valor del 100 %, lo cual nos dice que no tiene falsos positivos dentro de su clasificación, no se considera la especificidad alta que se tiene en la etapa 1 y 2 del cuarto modelo ya que sabemos que no tiene nigún falso positivo pero además tampoco tiene ningún verdadero positivos, es decir ninguna clasifiación correcta en esas etapas.

Figura 51

Comparación de Medida F1 en modelos de ANN

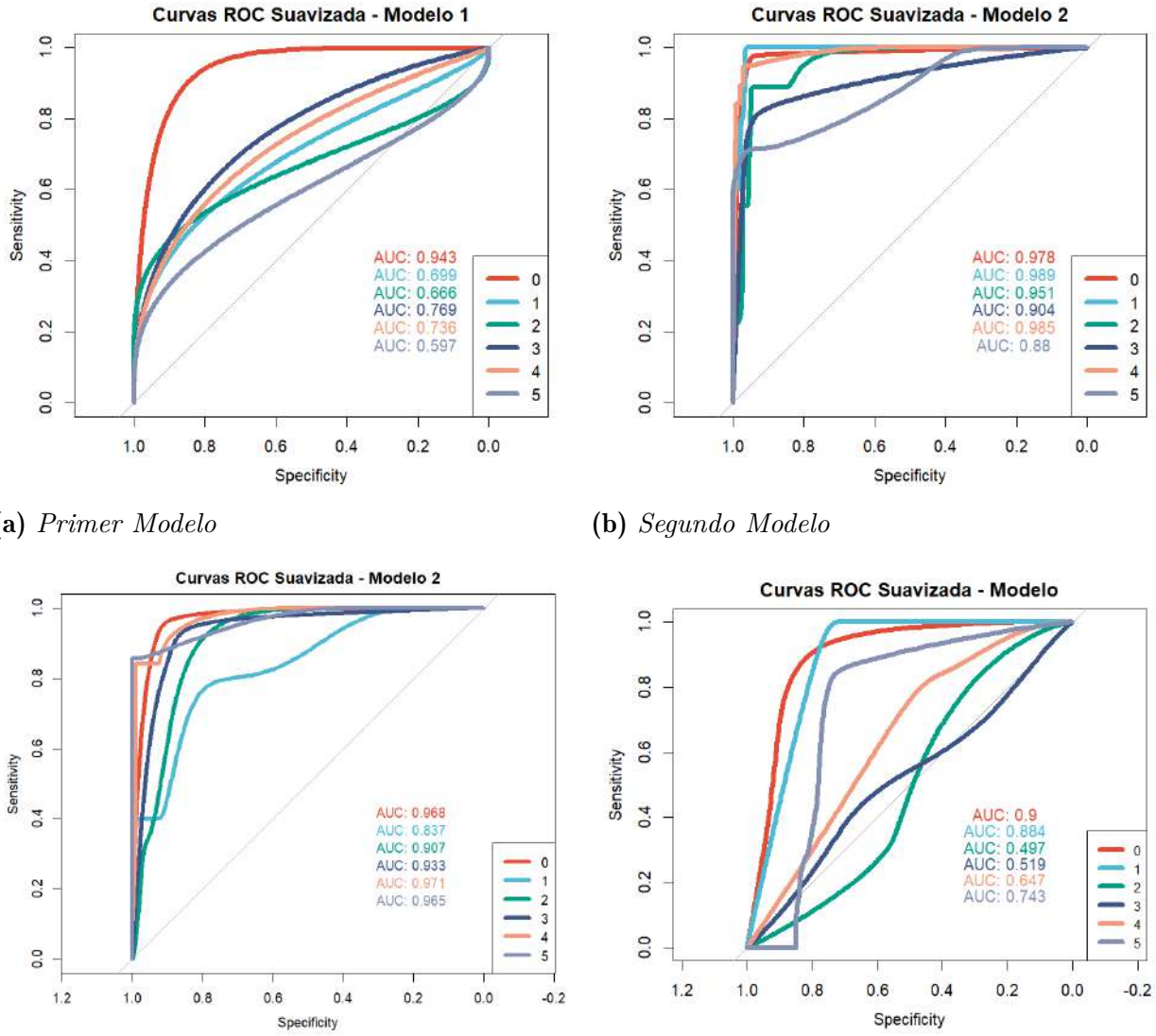


Nota: Medida F1 en % para los cuatro modelos de ANN. Elaborado por: (El autor, 2024).

En el caso de la medida F1 debemos fijarnos en el que tenga un nivel más alto por cada una de las etapas, ya que de esta forma nos aseguraremos de reducir la cantidad de falsos positivos y la de falsos negativos. De todos los modelos el que tiene a este métrica con buenos valores para la mayoría de sus estapas es el modelo 2, ya que tanto para la etapa 0, 2, 3 4 y 5 tiene un nivel de F1 por encima del 70 % , es decir a la vez que tiene una cantidad baja de falsos negativos también las tendrá en falsos potivos, donde no tiene un buen valor en su etapa 1 ya que tiene un valor del 44 %. En cambio en los demás modelos tienen en sus etapas 1 y 2 métricas por deabjo del 70 %, siendo el modelo 5 el que peor rendimiento tiene ya que es del 0 %.

Figura 52

Comparación curva ROC y valor AUC en modelos ANN



(a) *Primer Modelo*

(b) *Segundo Modelo*

(c) *Tercer Modelo*

(d) *Cuarto Modelo*

Nota: Elaborado por: (El autor, 2024).

Dentro de las variantes de las ANN, los dos únicos modelos que tienen un valor confiable de AUC en la curva ROC es el segundo y el tercer modelo. Como se observa en la figura 53 (b) y (c), el segundo modelo tiene un mejor desempeño en esta métrica en todas las etapas, ya que tiene un valor por encima del 80 %, en cambio en el modelo que usa tan solo dos características, empieza a perder desempeño para las etapas 2 y 3. Es por esto que el segundo

modelo es mejor.

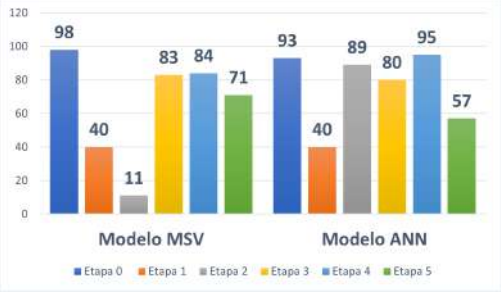
7.3. Comparación de los mejores modelos de redes neuronales con máquinas de soporte vectorial

Figura 53

Comparación de mejor modelo de SVM con ANN

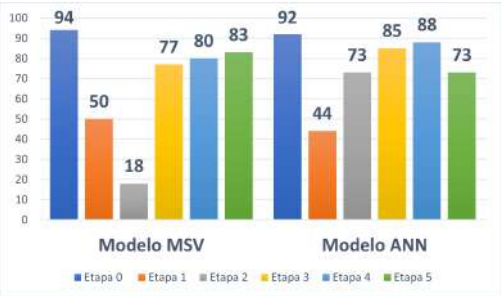
(a) *Accuracy*

(b) *Precisión*



(c) *Sensibilidad*

(d) *Especificidad*



(e) *Medida F1*

Nota: En el mejor modelo de SVM se uso tan solo 2 características, en cambio en el mejor de ANN se hizo uso de 6. Elaborado por: (El autor, 2024).

Para poder obtener el modelo que mejor se comportó para los datos de prueba hay que basarse en las métricas de evaluación de un modelo de clasificación, las cuales fueron observadas en varias etapas de la ERC, donde se comparó al mejor modelo de las ANN con el mejor modelo de las SVM, de esta forma se obtuvo las gráficas de la figura 53.

Para la precisión en este caso es de gran importancia tener la precisión más alta posible para las etapas 0 y la 5, ya que de esa forma se asegurará la disminución de instancias mal clasificadas en el caso de las personas sanas y en el caso de las personas que tienen ERC en etapa 5. Al comparar los dos modelos, se observa que en el modelo de ANN, en la etapa 0, logró un aumento del 1 %, alcanzando una precisión del 91 %. Posteriormente, en la etapa 2, se registró un notable incremento del 11 %, alcanzando una precisión del 61 %. La etapa 3 experimentó una mejora del 4 %, elevando la precisión a un 76 %, mientras que en la etapa 4 se observó un avance del 6 %, logrando una precisión del 82 %. En la etapa 5, el rendimiento alcanzó el 100 %, igualando el desempeño del modelo de SVM.

Sin embargo, en la etapa 1, se evidenció una disminución del 17 % en la precisión del modelo de ANN en comparación con el 67 % presentado por el modelo de SVM. Esto indica una reducción notable en la eficacia de la identificación en la etapa 1, siendo la única instancia donde el modelo de SVM superó al de ANN. Dado que en la mayoría de las etapas, el modelo de ANN logra una mejora por precisión de cada etapa con respecto al modelo de SVM, en esta métrica gana el modelo de ANN.

La sensibilidad también es una métrica en la que se debe buscar su máximo valor, preferiblemente en todas las etapas, pero es especialmente crucial para las etapas 0 y 5. De esta manera, se asegura que la mayoría de las personas, tanto las que realmente no tienen ERC como las que sí la padecen en la etapa 5, sean clasificadas correctamente. El modelo de SVM es el que presenta una mejor sensibilidad para la mayoría de las etapas. En la etapa 0, 3, 4 y 5, se observan valores del 98 %, 83 %, 84 % y 71 %, respectivamente. Sin embargo, su rendimiento no es óptimo en las etapas 1 y 2, con valores del 40 % y 11 %, respectivamente, lo que representa una proporción significativa de falsos negativos en estas fases.

En cambio en el modelo de ANN se tiene una buena sensibilidad para las etapas 0, 2, 3 y 4 ya que tienen un valor del 93 %, 89 %, 80 % y 95 %, respectivamente. En este modelo hubo una reducción del 5 % para la etapa 0, 3 % para la etapa 3 y un 14 % para la etapa 5. Y en el caso de la etapa 1, el 40 % se mantiene al igual que en el modelo de SVM. Debido a la

reducción del rendimiento del modelo de SMV para la etapa 1 y la etapa 2, el mejor modelo sería el de ANN.

Para la especificidad se buscará de igual forma la más alta para poder identificar correctamente a los casos negativos, sobre todo esta métrica es de importancia para la minimización de los falsos positivos en cada etapa. Para los modelos de MSV la mayoría de las etapas tienen una sensibilidad por encima del 90 %, pero tan solo en la 3 etapa tiene un valor del 87 %. A diferencia del modelo de ANN en todas sus etapas se tiene una especificidad por encima del 90 %, lo cual es bueno ya que minimizará la cantidad de instancias mal clasificadas en todas sus etapas, por esta razón en términos de sensibilidad el mejor modelo fue el de ANN.

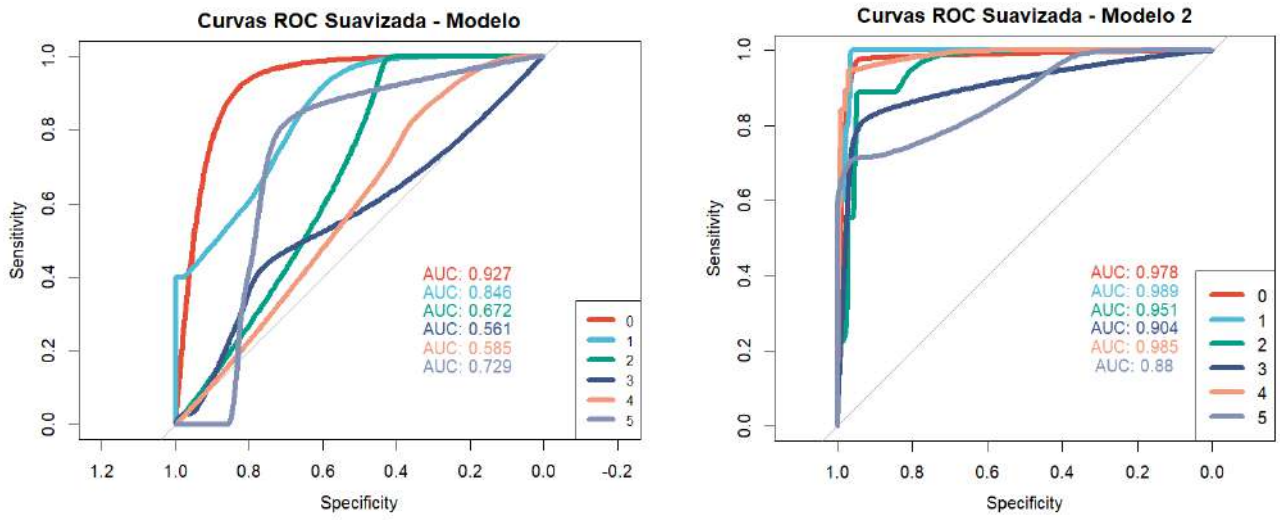
En este caso es de relevancia que se tengan las mejores métricas entre precisión y sensibilidad, es decir de lo que clasifique el modelo, tener una aumento en la cantidad de aciertos, y de lo que realmente pertenezca a una etapa, que el modelo pueda clasificarlo sin dificultad. Para esto se toma en cuenta la medida F1.

Por ejemplo, en el modelo de SVM, cuatro etapas exhiben un rendimiento sólido con valores superiores al 70 %. Sin embargo, en las etapas 1 y 2, se observan valores más bajos del 50 % y 18 %, respectivamente. Estos bajos valores sugieren la posibilidad de un aumento significativo en los falsos positivos y falsos negativos en estas dos etapas.

En contraste, el modelo de ANN abarca cinco etapas, todas con valores superiores al 70 %. Destaca la mejora sustancial en la etapa 2, que pasa de un rendimiento del 18 % a un 73 %. No obstante, se observa una reducción del 6 % en el rendimiento de la etapa 1 y, especialmente, reducciones del 2 % y 10 % en las etapas 0 y 5, respectivamente.

Figura 54

Comparación curva ROC y valor AUC de mejor modelo de SVM con ANN



(a) *Tercer Modelo - SVM*

(b) *Segundo Modelo - ANN*

Nota: Elaborado por: (El autor, 2024).

Considerando la métrica AUC de cada una de las etapas tanto para el modelo de SVM y el de ANN. En todas las etapas del modelo de ANN se tiene un valor de AUC por encima del 80 %, lo cual significa que este modelo realmente clasifica las diferentes etapas, de forma que por cada etapa reduce la cantidad de falsos positivos en sus clasificaciones. En cambio el modelo de SVM apesar de que tenía métricas similaes en algunas etapas, tiene un valor de AUC por debajo del 80 % tanto para la etapa 2, 3, 4 y 5, por lo que este modelo no es confiable para predecir estas etapas.

8. CONCLUSIONES, RECOMENDACIONES Y TRABAJOS A FUTURO

8.1. Conclusiones

- Como resultado final, se determinó que la utilización de las ANN fue la opción más efectiva. Ya que era capaz de clasificar a instancias de prueba que no padecían ERC, con

una precisión del 91 % y una sensibilidad del 93 %. A la misma vez que si se presentaban instancias con un fallo renal grave, es decir en etapa 5, las lograba clasificar con una precisión del 100 % y con una sensibilidad del 57 %. En el caso de personas con etapas 1 y 2, el modelo tuvo un rendimiento por debajo del 70 %, por lo cual no es fiable en estas etapas. En el caso de la etapa 3, el modelo llego a una precisión del 76 % y una sensibilidad del 80 %. En cambio, en la etapa 4, se tuvo una precisión del 82 % y una sensibilidad del 95 %.

- Una de las principales limitaciones evidenciadas por el modelo de ANN radicó en las métricas presentadas para las etapas 1 y 2. Para la etapa 1 generó una precisión del 50 %, con una sensibilidad del 40 %, y en el caso de la etapa 2 se obtuvo una precisión del 61 % con una sensibilidad del 89 %.

  Estás métricas podrían conllevar a que individuos que requieran de un tratamiento de tipo farmacológico a una dosis establecida por su etapa, reciban un tratamiento erróneo debido a la mala clasificación, incrementando de esta forma potencialmente el riesgo de mortalidad debido al avance de la enfermedad en sí. Además, esto podría llevar a un gasto y uso ineficiente de recursos limitados, que podrían haber sido destinados a personas verdaderamente necesitadas de ese tratamiento. Es por esto que es crucial diferenciar entre las diferentes etapas de la enfermedad, de forma que se garantice un diagnóstico preciso y a la vez una asignación eficiente de recursos en entornos donde la disponibilidad de recursos es crucial.

- El desarrollo del modelo final de clasificación automática presenta una perspectiva innovadora que sugiere su potencial aplicación como un medio adicional para el diagnóstico rápido de la ERC. Este enfoque adquiere especial relevancia al priorizar la correcta clasificación de individuos sin esta enfermedad y aquellos que la padecen, destacando especialmente las etapas 0, 3, 4 y 5. La capacidad del modelo para realizar clasificaciones precisas en estas fases específicas, ayuda a reducir la cantidad de falsos positivos o clasificaciones erróneas.

8.2. Recomendaciones

Al momento de realizar los modelos de SVM hay que tomar en cuenta dentro de los hiperparámetros tanto los kernels, el costo y la cantidad de vectores de soporte que se genera en el modelo, por que en el caso de usar un costo demasiado alto y tener como respuesta una cantidad de vectores de soporte muy cercana a los datos de entrenamiento, ya se podría estar hablando de algun tipo de sobre ajuste dentro del modelo.

En el caso de las redes neuronales hay que saber que a medida que generamos un modelo de alta complejidad o bien tenemos un modelo tan dificil que no logrará converger o bien que tendrá un desempeño perfecto que generará como resultado un sobre ajuste para nuevos datos que se ingresen al modelo. Es por esto que la complejidad del modelo dependerá directamente de las variables predictoras o variables que se usarán para el entrenamiento del modelo y debido a este será necesario que hagamos una experimentación computacional con las diferentes variantes en cuanto la cantidad de capas ocultas y el número de neuronas que tendra cada capa oculta.

Debido a la escasa cantidad de instancias utilizadas tanto en la etapa de entrenamiento como en la de prueba, especialmente en las etapas 1, 2 y 5, y con el objetivo de mejorar las métricas de evaluación en estas etapas, se recomienda como estrategia fundamental para potenciar el rendimiento del modelo la ampliación de la base de datos. La inclusión de un mayor número de instancias permitirá establecer un equilibrio más sólido entre los diversos datos de entrenamiento correspondientes a cada etapa de la ERC. Esta expansión de datos no solo enriquecerá la diversidad del conjunto de entrenamiento, sino que también fortalecerá la capacidad del modelo para generalizar patrones y mejorar su habilidad predictiva en nuevas instancias. Este enfoque se presenta como esencial para elevar las métricas de evaluación en las etapas mencionadas y, en última instancia, mejorar la robustez del modelo frente a la complejidad de los datos asociados con la ERC.

8.3. Trabajos a futuro

Para facilitar la interacción con el modelo, se propone el desarrollo de una interfaz gráfica intuitiva. Esta interfaz no solo permitiría ingresar el mejor modelo derivado de este trabajo, sino que también posibilitaría la evaluación de nuevas instancias. Los usuarios podrán ingresar información relativa a las características utilizadas por el mejor modelo, obteniendo así predicciones rápidas sobre la presencia o ausencia de ERC en los casos evaluados. El desarrolló de

esta interfaz abrirá nuevas posibilidades para la aplicación práctica de los resultados obtenidos, permitiendo realizar análisis más detallados y personalizados. Esta iniciativa contribuirá al avance de la investigación en el ámbito de la detección temprana de la ERC.

Referencias

Abdar, M., Zomorodi-Moghadam, M., Das, R., y Ting, I.-H. (2017). Performance analysis of classification algorithms on early detection of liver disease. *Expert Systems with Applications*, *67*, 239-251. doi: https://doi.org/10.1016/j.eswa.2016.08.065

AKF's. (2023). *Etapas o estadios de la enfermedad renal.* Recuperado de: `https://www.kidneyfund.org/es/todo-sobre-los-rinones/etapas-o-estadios-de-la-enfermedad-renal`.

Arce, J. I. B. (2019). *La matriz de confusión y sus métricas.* Recuperado de: `https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/`.

Aucejo, C. G. (2022). *Análisis de la robustez de los clasificadores dependiendo de la importancia de los atributos.* Universidad Politécnica de Valencia. Descargado de `https://riunet.upv.es/bitstream/handle/10251/188904/Galvez%20-%20Analisis%20de%20la%20robustez%20de%20los%20clasificadores%20dependiendo%20de%20la%20importancia%20de%20los%20atrib....pdf?sequence=1`

Belina, S., y K, K. (2018). Ensemble swarm behaviour based feature selection and support vector machine classifier for chronic kidney disease prediction. *International Journal of Engineering and Technology(UAE)*, *7*(2), 190-195.

Burballa, C., Crespo, M., Pachón, D. R., Sáez, M. J. P., Mir, M., Cabrales, C. A., ... Pascual, J. (2018). MDRD o CKD-EPI en la estimación del filtrado glomerular del donante renal vivo. *Nefrologia*, *38*(2), 207-212.

Diez, J. J. R. (2023). *Aprendizaje automÁtico en ciencia de datos.* Recuperado de: `https://www.ubu.es/sites/default/files/news/files/leccion_inaugural_2023-2024.pdf`.

Douketis, J. D. (2022). *Linfedema.* Recuperado de: `https://www.msdmanuals.com/es-ec/professional/trastornos-cardiovasculares/trastornos-linf%C3%A1ticos/linfedema`.

Evia, J. R. B. (2008). Marcadores de índice de filtración glomerular: Cistatina C. *Rev Mex Patol Clin*, *55*(3), 149-156.

García-Maset, R., Bover, J., de la Morena, J. S., Diezhandino, M. G., del Hoyo, J. C., Martín, J. E. S., ... Górriz, J. L. (2022). Information and consensus document for the detection and management of chronic kidney disease. *Nefrología*, *42*(3), 233-264.

Gupta, R., Kumari, S., Senapati, A., Ambasta, R. K., y Kumar, P. (2023). New era of artificial intelligence and machine learning-based detection, diagnosis, and therapeutics in Parkinson's disease. *Ageing Research Reviews*, *90*, 102013. doi: https://doi.org/

10.1016/j.arr.2023.102013

Guzmán, K., Fernández, J., Mora, F., y Vintimilla, J. (2014). Prevalence and risk factors for chronic renal disease. *Revista Médica Del Hospital General De México*, *77*(3), 108-113.

INEC. (2020). *Registro estadístico de defunciones generales de 2020*. Recuperado de: https://www.ecuadorencifras.gob.ec/documentos/web-inec/Poblacion_y_Demografia/Defunciones_Generales_2020/2021-06-10_Principales_resultados_EDG_2020_final.pdf.

INEC. (2022). *Camas y egresos hospitalarios*. Recuperado de: https://app.powerbi.com/view?r=eyJrIjoiMmE3NDMwOGMtZGJlOC00MDJhLWEwYWMtZDg1MmMwZmViNDBmIiwidCI6ImYxNThhMmU4I

KeepCoding. (2022). *Clasificación estadística de regresión logística*. KeepCoding Tech School. Descargado de https://keepcoding.io/blog/clasificacion-estadistica-regresion-logistica/

KYOCERA. (2023). *Qué es el aprendizaje automático*. Recuperado de: https://www.kyoceradocumentsolutions.es/es/smarter-workspaces/insights-hub/articles/que-es-el-aprendizaje-automatico.html.

Lamorte, J. M. (2023). *Todo lo que se necesita saber sobre el machine learning*. Recuperado de: https://www.itmastersmag.com/noticias-analisis/todo-lo-que-se-necesita-saber-sobre-el-machine-learning/.

Levey, A., Atkins, R., Coresh, J., Cohen, E., Collins, A., Eckardt, K.-U., ... Eknoyan, G. (2007). Chronic kidney disease as a global public health problem: Approaches and initiatives – a position statement from Kidney Disease Improving Global Outcomes. *Kidney International*, *72*(3), 247-259.

Levey, A. S., Stevens, L. A., Schmid, C. H., Zhang, Y. L., III, A. F. C., Feldman, H. I., ... Coresh, J. (2009). A new equation to estimate glomerular filtration rate. *Annals of Internal Medicine*, *150*(9), 604.

Liao, Y., Liao, W., Liu, J., Xu, G., y Zeng, R. (2011). Assessment of the ckd-epi equation to estimate glomerular filtration rate in adults from a chinese ckd population. *Journal of International Medical Research*, *39*, 2273 - 2280.

Malkina, A. (2023). *Enfermedad renal crónica o nefropatía crónica*. Recuperado de: https://www.msdmanuals.com/es/hogar/trastornos-renales-y-del-tracto-urinario/insuficiencia-renal/enfermedad-renal-cr%C3%B3nica-o-nefropat%C3%ADa-cr%C3%B3nica.

Martín de Francisco, A. L., C.Piñera, M.Gago, J.Ruiz, C.Robledo, y M.Arias. (2009). Epidemiología de la enfermedad renal crónica en pacientes no nefrológicos. *Nefrología*, *16*(5),

1-130.

Massaro, A., Galiano, A., Scarafile, D., Vacca, A., Frassanito, A., Melaccio, A., ... Bonomo, M. (2020). *Plataforma multinivel dss-ai de telemedicina para asistencia en gammapatía monoclonal.* Recuperado de: https://scholar.google.com/scholar_lookup?title=Telemedicine+DSS-AI+Multi+Level+Platform+for+Monoclonal+Gammopathy+Assistance&conference=Proceedings+of+the+2020+IEEE+International+Symposium+on+Medical+Measurements+and+Applications+(MeMeA)&author=Massaro,+A.&author=Galiano,+A.&author=Scarafile,+D.&author=Vacca,+A.&author=Frassanito,+A.&author=Melaccio,+A.&author=Solimando,+A.&author=Ria,+R.&author=Calamita,+G.&author=Bonomo,+M.&publication_year=2020&doi=10.1109/MEMEA49120.2020.9137224.

MatLab. (S.f). *Support vector machine (svm).* Recuperado de: https://la.mathworks.com/discovery/support-vector-machine.html.

Medina, F., y Galván, M. (2007). *Imputación de datos: teoría y práctica* (n.º 54). Unidad de Estadísticas Sociales de la División de Estadística y Proyecciones Económicas de la Comisión Económica para América Latina y el Caribe.

Mirzaei, G., y Adeli, H. (2022). Machine learning techniques for diagnosis of alzheimer disease, mild cognitive disorder, and other types of dementia. *Biomedical Signal Processing and Control*, *72*, 103293. doi: https://doi.org/10.1016/j.bspc.2021.103293

Mirzaei, G., y Adeli, H. (2023). Identification of Alzheimer's disease from central lobe EEG signals utilizing machine learning and residual neural network. *Biomedical Signal Processing and Control*, *86*, 105266. doi: https://doi.org/10.1016/j.bspc.2023.105266

Morales, V., y Ricardo, G. (2019). *Clasificador con redes neuronales para el pronóstico de la enfermedad renal crónica en la población colombiana.* Recuperado de: https://reunir.unir.net/handle/123456789/9446.

NIDDKD. (2018). *Cómo elegir un tratamiento para la insuficiencia renal.* Recuperado de: https://www.niddk.nih.gov/health-information/informacion-de-la-salud/enfermedades-rinones/insuficiencia-renal/como-elegir-tratamiento.

NKF. (2002). *Clinical practice guidelines for chronic kidney disease: Evaluation, classification and stratification.* Recuperado de: https://www.kidney.org/sites/default/files/docs/ckd_evaluation_classification_stratification.pdf.

Nowak, S. (2023). *Diferencias entre aprendizaje supervisado y no supervisado.* Recuperado de: https://nuclio.school/diferencias-aprendizaje-supervisado-y-no-supervisado/e.

Olivera, O. G. O. (2019). *Redes neuronales artificiales: Qué son y cómo se entrenan.*

Recuperado de: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i.

OPS. (2015). *La OPS/OMS y la Sociedad Latinoamericana de Nefrología llaman a prevenir la enfermedad renal y a mejorar el acceso al tratamiento.* Recuperado de: https://www3.paho.org/hq/index.php?option=com_content&view=article&id=10542:2015-opsoms-sociedad-latinoamericana-nefrologia-enfermedad-renal-mejorar-tratamiento&Itemid=0&lang=es#gsc.tab=0.

Persaud, C., Sandesara, U., Hoang, V., Tate, J., Latack, W., y Dado, D. (2021). *Creatinina sérica más alta registrada* (Vol. 2021).

Rady, E.-H. A., y Anwar, A. S. (2019). Prediction of kidney disease stages using data mining algorithms. *Informatics in Medicine Unlocked*, *15*, 100178.

Rodrigo Orozco, B. (2010). Prevención y tratamiento de la enfermedad renal crónica (erc). *Revista Médica Clínica Las Condes*, *21*(5), 779-789.

Rodriguez, C. C. (2020). *Maquina de soporte vectorial (svm).* Recuperado de: https://medium.com/@csarchiquerodriguez/maquina-de-soporte-vectorial-svm-92e9f1b1b1ac.

ROYO, D. F. J. L. (2022). *Edemas.* Recuperado de: https://www.cun.es/enfermedades-tratamientos/enfermedades/edemas.

Santos, P. R. D. L. (2021). *Tipos de aprendizaje en machine learning: supervisado y no supervisado.* Recuperado de: https://telefonicatech.com/blog/que-algoritmo-elegir-en-ml-aprendizaje.

Sellarés, V. L., y Rodríguez, D. (2023). Enfermedad Renal Crónica. *Nefrología al día.*

Shahabeddin, A., R., N. K. S., Mehdi, E., Hajar, H., y Ali, G. (2019). Artificial Intelligence Applications in Type 2 Diabetes Mellitus Care: Focus on Machine Learning Methods. *Healthc Inform Res*, *25*(4), 248-261. doi: 10.4258/hir.2019.25.4.248

Simeone, O. (2018). A very brief introduction to machine learning with applications to communication systems. *IEEE Transactions on Cognitive Communications and Networking*, *4*(4), 648-664.

Sánchez, B. L. P., Guacho, J. S. G., y Guerrero, I. R. M. (2021). Chronic kidney disease. Literature review The local experience in an Ecuador city. *Revista Colombiana de Nefrologia*, *8*(3), 233-264.

UCIrvine. (2015). *Chronic kidney disease.* Recuperado de: https://archive.ics.uci.edu/dataset/336/chronic%2Bkidney%2Bdisease.

Verdejo, H., Díaz, F., Castro, P., Rossef, V., Concepción, R., y Sepúlveda, L. (2014). Estimación indirecta de la función renal y mortalidad por insuficiencia cardíaca: buscando el mejor

predictor. *Revista chilena de cardiología*, *33*(3), 189-197.

Vidal, J., y Vidal, O. (2022). Aplicaciones de la inteligencia artificial en la medicina. *Revista Peruana de Investigación En Salud*, *6*(3), 131-133.

Vilche, J., Alejandro, M., y Correa, V. (2022). Diferencias entre las ecuaciones mdrd4-idms y ckd-epi 2009: significancia estadística y clínica. *Bioquímica y Patología Clínica*, *86*(2), 36-42.

Villegas, B. F., Lazcano, I. F., y de Lourdes Lazcano Mendoza, M. (2014). Edema. enfoque clínico. *Med Int Méx*, *30*(1), 51-55.

Xie, G., Chen, T., Li, Y., Chen, T., Li, X., y Liu, Z. (2019). Artificial Intelligence in Nephrology: How Can Artificial Intelligence Augment Nephrologists' Intelligence? *Kidney Diseases*, *6*(1), 1-6.

ANEXOS

Anexo 1: Limpieza y Preprocesamiento de los datos

Importación de liberarías

```
library(readr)
library(openxlsx)
library(dplyr)
library(VIM)
```

Carga de la base de datos totalmente cruda

```
Base_de_datos_CKD <- read_csv("C:/Users/alext/Desktop/P_Tesis/
   Base␣de␣datos/Chronic_Kidney_Disease/Base_de_datos_CKD.txt")
```

Limpieza de datos faltantes y correción de datos atípicos

```
# Correcion de datos atipicos en variables nominales.
Base_de_datos_CKD$appet=ifelse(Base_de_datos_CKD$appet=="no","
   poor",Base_de_datos_CKD$appet)
Base_de_datos_CKD$pe=ifelse(Base_de_datos_CKD$pe=="good","?",
   Base_de_datos_CKD$pe)
Base_de_datos_CKD$classification=ifelse(Base_de_datos_CKD$
   classification=="ckd,","ckd",Base_de_datos_CKD$classification
   )
Base_de_datos_CKD$classification=ifelse(Base_de_datos_CKD$
   classification=="no,notckd","notckd",Base_de_datos_CKD$
   classification)

# Verificacion datos faltantes en todas las variables de la
   data.
Base_de_datos_CKD[Base_de_datos_CKD == "?"] <- "Faltante"
Base_de_datos_CKD[Base_de_datos_CKD == "NA"] <- "Faltante"
```

```
# Total de datos faltantes
datos_faltantes<- sum(apply(Base_de_datos_CKD, MARGIN = 1,
   function(x) sum(x == "Faltante", na.rm = TRUE)))

# Reemplaza "Faltante" con NA en la columna "columna_faltante"
Base_de_datos_CKD[Base_de_datos_CKD == "Faltante"] <- NA

# Redondeamos la edad para transformar algunos datos que sean
   cuantitativos continuo a discreto
Base_de_datos_CKD$age=round(Base_de_datos_CKD$age)
```

Conversión a factor a variables nominales y númericas las cuantitativas

```
# -- Conversion a factor a variables nominales y numericas las
   cuantitativas

numericr_columns_1 <- lapply(Base_de_datos_CKD[, 1:2], as.
   numeric)
factor_columns_1 <- lapply(Base_de_datos_CKD[, 3:9], as.factor)
numeric_columns_2<- lapply(Base_de_datos_CKD[, 10:18], as.
   numeric)
factor_columns_3 <- lapply(Base_de_datos_CKD[, 19:25], as.
   factor)

# Combinar las variables convertidas
Base_de_datos_CKD<- cbind.data.frame(numericr_columns_1 ,
   factor_columns_1, numeric_columns_2,factor_columns_3)
```

Distribución de los datos despues de la limpieza

```
# Separamos dos dataframes: Uno para datos categoricos y otro
   para cuantitativos
```

```
Categoric_data=cbind(Base_de_datos_CKD[,3:9],Base_de_datos_CKD
   [,19:25])
Numeric_data=cbind(Base_de_datos_CKD[,1:2],Base_de_datos_CKD
   [,10:18])

#---------- Visualizacion de la distribucion de los datos

#----------- Diagramas de caja y bigote
# Ruta donde se guardaran las imagenes
ruta_diagrama_caja_bigote<- "C:/Users/alext/Desktop/P_Tesis/
   Imagenes/Diagramas de Bigotes y cajas"

# Genera y guarda los diagramas de caja y bigotes
for (i in names(Numeric_data)) {
Numeric_data_no_na=na.omit(Numeric_data[[i]])

bp <- boxplot(Numeric_data_no_na, main = paste("Diagrama de
   Caja y Bigotes ",i) , ylab = "Valores", labels = FALSE,col =
   "lightblue")

mediana_valor <- median(Numeric_data_no_na,na.rm = TRUE)
cuartil_1_valor <- quantile(Numeric_data_no_na,na.rm = TRUE,
   0.25)
cuartil_3_valor <- quantile(Numeric_data_no_na,na.rm = TRUE,
   0.75)
media_valor <- mean(Numeric_data_no_na,na.rm = TRUE)
sd_valor <- sd(Numeric_data_no_na,na.rm = TRUE)

legend("topright", legend = c("Mediana", "Q1", "Q3", "Media"),
   col = c("green", "blue", "blue", "red"), lty = 1, cex = 0.6)

abline(h = mediana_valor, col = "green", lty = 2)
abline(h = cuartil_1_valor, col = "blue", lty = 2)
abline(h = cuartil_3_valor, col = "blue", lty = 2)
```

```
abline(h = media_valor, col = "red", lty = 2)

title(main = paste("Mediana:", round(mediana_valor, 1),
                   "␣␣␣Q1:", round(cuartil_1_valor, 1),
                   "␣␣␣Q3:", round(cuartil_3_valor, 1),
                   "␣␣␣Media:", round(media_valor, 1)),
                   sub = "", line = -26, cex.main = 1.7, cex.sub =
                       1.2)

# Guarda el diagrama como archivo PNG en la carpeta especifica
filename <- file.path(ruta_diagrama_caja_bigote, paste0(i, ".
   png"))
dev.copy(png, filename)
dev.off()
}

#---------------- Campana de Gauss
# Ruta donde se guarda la imagen
ruta_campana_gauss<- "C:/Users/alext/Desktop/P_Tesis/Imagenes/
   Campanas␣de␣Gauss/"

for (i in names(Numeric_data)) {

Numeric_data_no_na= na.omit(Numeric_data[[i]])

media <- mean(Numeric_data_no_na,na.rm = TRUE)
mediana <- median(Numeric_data_no_na,na.rm = TRUE)
desviacion_estandar <- sd(Numeric_data_no_na,na.rm = TRUE)

# Crear histograma y superponer la curva de densidad normal
histo=hist(Numeric_data_no_na, probability = TRUE,
col = "lightblue",
main = paste("Histograma␣y␣Curva␣de␣Densidad␣Normal",i ,"\
   nMedia␣", round(media, 2), "␣Mediana␣=", round(mediana, 2), "
```

```
   ␣Desviacion␣Estandar␣=",round(desviacion_estandar, 2)), xlab
   = "Valor",ylab = "Densidad␣de␣Probabilidad",cex.main = 0.8)

x <- seq(min(Numeric_data_no_na), max(Numeric_data_no_na),
   length=100)
y <- dnorm(x, mean = media, sd = desviacion_estandar)
lines(x, y, col = "blue", lwd = 2)

# Guardar el diagrama como archivo PNG en la carpeta especifica
filename <- file.path(ruta_campana_gauss, paste0(i, ".png"))
dev.copy(png, filename)
dev.off()
}
```

Identificación y tratamiento de outliers

```
# Identificar y tratar outliers por cada variable
for (col in colnames(Numeric_data)) {
# Calcular el rango intercuartilico (IQR) excluyendo NAs
iqr <- IQR(Numeric_data[[col]], na.rm = TRUE)

# Definir limites para identificar outliers excluyendo NAs
lower_limit <- quantile(Numeric_data[[col]], 0.25,
na.rm = TRUE) - 1.5 * iqr
upper_limit <- quantile(Numeric_data[[col]], 0.75,
na.rm = TRUE) + 1.5 * iqr

outliers=Numeric_data[[col]] < lower_limit | Numeric_data[[col
   ]] > upper_limit

Numeric_data[[col]] <- ifelse(outliers & !is.na(Numeric_data[[
   col]]), median(Numeric_data[[col]], na.rm = TRUE), Numeric_
   data[[col]]) }
```

Imputación de los datos

```
k_valor <- 4

datos_imputados_numeric <- kNN(Numeric_data,  k = k_valor,
   metric = "euclidean")
datos_imputados_categoric<- kNN(Categoric_data, k = k_valor,
   metric = "euclidean")

datos_imputados_numeric=datos_imputados_numeric[,1:11]
```

Distribución de los datos despues del tratamiento de outliers y la immputación

```
#---------------- Campana de Gauss
ruta_campana_gauss_imput="C:/Users/alext/Desktop/P_Tesis/
   Imagenes/Campana de Gauss - Despues de imputacion/"

for (i in names(datos_imputados_numeric)) {

  media <- mean(datos_imputados_numeric [[i]],na.rm = TRUE)
  mediana <- median(datos_imputados_numeric [[i]],na.rm = TRUE)
  desviacion_estandar <- sd(datos_imputados_numeric [[i]],na.rm
      = TRUE)

  histo=hist(datos_imputados_numeric [[i]], probability = TRUE,
             col = "lightblue",
             main = paste("Histograma y Curva de Densidad
                Normal",
                          i ,"\nMedia ", round(media, 2), "
                             Mediana =",
                          round(mediana, 2), " Desviacion
                             Estandar =",
```

```
                              round(desviacion_estandar, 2)), xlab
                                 = "Valor",
                              ylab = "Densidad de Probabilidad",cex
                                 .main = 0.8)

  x <- seq(min(datos_imputados_numeric [i]),
           max(datos_imputados_numeric [i]),length=100)
  y <- dnorm(x, mean = media, sd = desviacion_estandar)

  lines(x, y, col = "blue", lwd = 2)

  filename <- file.path(ruta_campana_gauss_imput, paste0(i, ".
     png"))
  dev.copy(png, filename)
  dev.off()

}

#---------------- Diagrama de caja y bigote
ruta_diagrama_caja_bigote_imput="C:/Users/alext/Desktop/P_Tesis
   /Imagenes/Diagrama de Bigotes y cajas - Despues de imputacion
   /"

for (i in names(datos_imputados_numeric)) {
  bp <- boxplot(datos_imputados_numeric[i],
                main = paste("Diagrama de Caja y Bigotes ",i),
                ylab = "Valores", labels = FALSE,col = "
                   lightblue")

  mediana_valor <- median(datos_imputados_numeric[[i]],na.rm =
     TRUE)
  cuartil_1_valor <- quantile(datos_imputados_numeric[[i]],na.
     rm = TRUE, 0.25)
  cuartil_3_valor <- quantile(datos_imputados_numeric[[i]],na.
```

```
    rm = TRUE, 0.75)
  media_valor <- mean(datos_imputados_numeric[[i]],na.rm = TRUE
    )
  sd_valor <- sd(datos_imputados_numeric[[i]],na.rm = TRUE)

  legend("topright", legend = c("Mediana", "Q1", "Q3", "Media")
    ,
         col = c("green", "blue", "blue", "red"), lty = 1, cex
           = 0.6)

  abline(h = mediana_valor, col = "green", lty = 2)
  abline(h = cuartil_1_valor, col = "blue", lty = 2)
  abline(h = cuartil_3_valor, col = "blue", lty = 2)
  abline(h = media_valor, col = "red", lty = 2)

  title(main = paste("Mediana:", round(mediana_valor, 2),
                     "␣␣␣Q1:", round(cuartil_1_valor, 2),
                     "␣␣␣Q3:", round(cuartil_3_valor, 2),
                     "␣␣␣Media:", round(media_valor, 2)),
        sub = "", line = -26)

  filename <- file.path(ruta_diagrama_caja_bigote_imput, paste0
    (i, ".png"))
  dev.copy(png, filename)
  dev.off()
}
```

Análisis de los datos

```
# -- Distribucion de los datos cuantitativos en dos dimensiones
#-- Escalamiento Multidimensional en variables cuantitativas

# Distribucion datos sin escalar ni estandarizar.
```

```
# Matriz de distancias
d1= dist(datos_imputados_numeric,method = "euclidean",diag =
   TRUE, upper = TRUE)
# Escalamiento Multidimensional
fit1= cmdscale(d1,k=2,eig = TRUE)
x = fit1$points[,1]
y = fit1$points[,2]
plot(x,y)

plot(x,y,col=c("green","red")[Base_de_datos_CKD$classification
   ],pch = 16)
title(main = "Distribucion de los datos")

#----------------------Estandarizacion
Estand_New_dat_numeric= scale(datos_imputados_numeric)
Estand_New_dat_numeric= as.data.frame(Estand_New_dat_numeric)
# Matriz de distancias
d3= dist(Estand_New_dat_numeric,method = "euclidean",diag =
   TRUE, upper = TRUE)
# Escalamiento Multidimensional
fit3= cmdscale(d3,k=2,eig = TRUE)
xxa = fit3$points[,1]
yyb = fit3$points[,2]
plot(xxa,yyb)
plot(xxa,yyb,col=c("green","red")[Base_de_datos_CKD$
   classification],pch = 16)
title(main = "Distribucion de los datos estandarizados")


#////////////////////////Cantidad de personas con ERC.
categorias <- c("Si ERC", "No ERC")

color_azul_crema <- "#ADD8E6"
color_verde_crema <- "#98FB98"
```

```
cantidades <- table(Base_de_datos_CKD$classification)
barplot_valores <- barplot(cantidades, names.arg = categorias,
                           col = c(color_azul_crema, color_
                              verde_crema),
                           main = "Enfermedad␣Renal␣Cronica",
                           xlab = "Categorias", ylab = "
                              Incidencia")

text(x = barplot_valores,
     y = cantidades + 1,
     labels = as.character(cantidades),
     pos = 1.5, cex = 1.2, col = "black")
#////////////////////// Edades

hist(datos_imputados_numeric$age, main = "Histograma␣-␣Edades",
     xlab = "Edades", ylab = "Frecuencia", labels = TRUE)

text(x = barplot_valores,
     y = cantidades + 1,
     labels = as.character(cantidades),  # Convertir a
        caracteres para evitar nombres
     pos = 1.5, cex = 1.2, col = "black")

#////////////////////// Hipertension
categorias <- c("No", "Si")

color_azul_crema <- "#ADD8E6"
color_verde_crema <- "#98FB98"
# Crear el grafico de barras
cantidades <- table(datos_imputados_categoric$htn)
barplot_valores <- barplot(cantidades, names.arg = categorias,
                           col = c(color_azul_crema, color_
                              verde_crema),
```

```
                              main = "Hipertension␣Arterial",
                              xlab = "Categorias", ylab = "
                                 Incidencia")

text(x = barplot_valores,
     y = cantidades + 1,
     labels = as.character(cantidades),
     pos = 1.5, cex = 1.2, col = "black")

#////////////////////// Diabetes
categorias <- c("No", "Si")
# 263 ; 137
color_azul_crema <- "#ADD8E6"
color_verde_crema <- "#98FB98"
# Crear el grafico de barras
cantidades <- table(datos_imputados_categoric$dm)
barplot_valores <- barplot(cantidades, names.arg = categorias,
                              col = c(color_azul_crema, color_
                                 verde_crema),
                              main = "Diabetes␣Mellitus",
                              xlab = "Categorias", ylab = "
                                 Incidencia")

text(x = barplot_valores,
     y = cantidades + 1,
     labels = as.character(cantidades),
     pos = 1.5, cex = 1.2, col = "black")

#////////////////////// ECV
categorias <- c("No", "Si")
# 366 ; 34
color_azul_crema <- "#ADD8E6"
color_verde_crema <- "#98FB98"
cantidades <- table(datos_imputados_categoric$cad)
```

```
barplot_valores <- barplot(cantidades, names.arg = categorias,
                           col = c(color_azul_crema, color_
                              verde_crema),
                           main = "Enfermedad␣de␣las␣arterias␣
                              coronarias",
                           xlab = "Categorias", ylab = "
                              Incidencia")

text(x = barplot_valores,
     y = cantidades + 1,
     labels = as.character(cantidades),  # Convertir a
        caracteres para evitar nombres
     pos = 1.5, cex = 1.2, col = "black")

#////////////////////////////ANEMIA
categorias <- c("No", "Si")
# 366 ; 34
color_azul_crema <- "#ADD8E6"
color_verde_crema <- "#98FB98"
cantidades <- table(datos_imputados_categoric$ane)
barplot_valores <- barplot(cantidades, names.arg = categorias,
                           col = c(color_azul_crema, color_
                              verde_crema),
                           main = "Anemia", xlab = "Categorias"
                              ,
                           ylab = "Incidencia")

text(x = barplot_valores,
     y = cantidades + 1,
     labels = as.character(cantidades),
     pos = 1.5, cex = 1.2, col = "black")
```

Procesamiento de los datos

```
# Aumento de variables
# ------------- Variables: Genero, TFGe
# Genero se aumenta.
datos_imputados_categoric=datos_imputados_categoric[,1:14]
set.seed(1)
cont_pe=sum(datos_imputados_categoric$pe =="yes")
porcentaje_87_yes <- round(cont_pe * 0.87)
Gen=rep(0, length(datos_imputados_categoric$pe))
Gen[sample(which(datos_imputados_categoric$pe == "yes"),
           porcentaje_87_yes)] = 1
Gen=as.factor(Gen)

datos_imputados_categoric=cbind(datos_imputados_categoric,Gen)
Data_CKD=cbind(datos_imputados_categoric,datos_imputados_numeric)

# TFGe
FGe=c()

for (i in 1:dim(Data_CKD)[1]) {

  if (Data_CKD$Gen[i] ==1 ) {
    if (Data_CKD$sc[i]<=0.7){
      FGe[i]=144*((Data_CKD$sc[i]/0.7)^-0.329)*(0.993)^Data_CKD$age[i]
    }
    else{
      FGe[i]=144*((Data_CKD$sc[i]/0.7)^-1.209)*(0.993)^Data_CKD$age[i]
    }
  }
  else{
    if (Data_CKD$sc[i]<=0.9){
      FGe[i]=141*((Data_CKD$sc[i]/0.9)^-0.411)*(0.993)^Data_CKD$age[i]
    }
    else{
      FGe[i]=141*((Data_CKD$sc[i]/0.9)^-1.209)*(0.993)^Data_CKD$age[i]
```

```
    }
  }
}

Data_CKD=cbind(Data_CKD[,1:14],FGe,Data_CKD$Gen,Data_CKD[,16:26])

# ANALIZANDO TFGE
hist(Data_CKD$FGe, main = "Histograma - TFGe",
     xlab = "Filtrado Glomerular", ylab = "Frecuencia", labels
        = TRUE)

# Medidas de tendencia central y dispersion en la TFGe
summary(Data_CKD$FGe)
etapa=c()
for (i in 1:dim(Data_CKD)[1]) {

  if (Data_CKD$FGe[i] >=90) {
    etapa[i]="1"
  }
  else{
    if (Data_CKD$FGe[i] >=60 & Data_CKD$FGe[i] <=89) {
      etapa[i]="2"
    }
    else{
      if (Data_CKD$FGe[i] >=30 & Data_CKD$FGe[i] <=59) {
        etapa[i]="3"
      }
      else{
        if (Data_CKD$FGe[i] >=15 & Data_CKD$FGe[i] <=29) {
          etapa[i]="4"
        }
        else{
          etapa[i]="5"
        } } } }}
```

```
datos_imputados_numeric=cbind(datos_imputados_numeric,FGe)
Data_B_CKD=cbind(Data_CKD,etapa)
Datt=c()

for (i in 1:dim(Data_B_CKD)[1]) {
  if (Data_B_CKD$classification[i] == "notckd") {
    Datt[i]="0"
  }
  else{
    Datt[i]=Data_B_CKD$etapa[i]
  }
}

etapa=Datt
Data_B_CKD=cbind(Data_CKD,etapa,Datt)
Data_B_CKD$age=round(Data_B_CKD$age)
Data_B_CKD=Data_B_CKD[,1:28]

# Analizando Etapa
# Datos de ejemplo
categorias <- c("Etapa 0", "Etapa 1", "Etapa 2",
                "Etapa 3", "Etapa 4", "Etapa 5")
frecuencias <- table(Data_B_CKD$etapa)

library(paletteer)
colores = paletteer_d("ggsci::nrc_npg")

barplot(frecuencias, names.arg = categorias, col = colores,
        main = "Gráfico de Barras - Etapa", ylab =
          "Frecuencia")

text(x = barplot(frecuencias, col = "black", plot = FALSE),
     y = frecuencias + 1, labels = frecuencias, pos = 1.5,
```

```
     cex = 0.8, col = "black")

write.csv(Data_B_CKD, file =
   "C:/Users/alext/Desktop/P_Tesis/Base de
   datos/Data_modif.csv", row.names = FALSE)

hist(Data_B_CKD$sc, main = "Histograma - Creatinina Sérica",
     xlab = "Creatinina", ylab = "Frecuencia", labels = TRUE)

hist(Base_de_datos_CKD$sc, main = "Histograma - Creatinina
   Sérica",
     xlab = "Creatinina", ylab = "Frecuencia", labels = TRUE)
```

Anexo 2: Código modelos SVM

Importación de liberarías

```
library(readr)
library(corrplot)
library(caTools)
library(caret)
library(e1071)
library(yardstick)
library(ggplot2)

# Aumento de variables
```

Carga de la base de datos procesada

```
Data_modif <- read_csv("C:/Users/alext/Desktop/P_Tesis/Base de
   datos/Data_modif.csv")
```

Conversión a factor a variables nominales y númericas las cuantitativas

```
factor_columns_1 <- lapply(Data_modif[, 1:14], as.factor)
numeric_columns_1<- lapply(Data_modif[, 15], as.numeric)
factor_columns_2 <- lapply(Data_modif[, 16], as.factor)
numeric_columns_2<- lapply(Data_modif[, 17:27], as.numeric)
factor_columns_3 <- lapply(Data_modif[, 28], as.factor)

# Combinar las variables convertidas
Data_modif<- cbind.data.frame(factor_columns_1,
   numeric_columns_1, factor_columns_2,
   numeric_columns_2,factor_columns_3)
```

Selección de características

```
mat_cor=cor(cuantitative_ckd)
pairs(mat_cor.1,upper.panel = NULL,col="blue",lower.panel =
   panel.smooth)
corrplot(mat_cor)

corrplot(mat_cor, method = "circle", type = "lower", order =
   "original",tl.cex = 1.8, cl.cex = 1.8)
pairs(mat_cor,upper.panel = NULL,col="blue",lower.panel =
   panel.smooth)

# Correlacion de variables con TFGe
# Mayor igual 0.5 E, US, CS, H, VCE y CR.
# Mayor igual 0.6 CS y VCE
# Mayor igual 0.7 CS
```

Modelo con todas las variables

```
#Cuali One - Hot - Encoding
```

```
Data_modif$sg=model.matrix(~ Data_modif$sg - 1, data =
   Data_modif)
Data_modif$al=model.matrix(~ Data_modif$al - 1, data =
   Data_modif)
Data_modif$su=model.matrix(~ Data_modif$su - 1, data =
   Data_modif)
Data_modif$rbc=model.matrix(~ Data_modif$rbc - 1, data =
   Data_modif)
Data_modif$pc=model.matrix(~ Data_modif$pc - 1, data =
   Data_modif)
Data_modif$pcc=model.matrix(~ Data_modif$pcc - 1, data =
   Data_modif)
Data_modif$ba=model.matrix(~ Data_modif$ba - 1, data =
   Data_modif)
Data_modif$htn=model.matrix(~ Data_modif$htn - 1, data =
   Data_modif)
Data_modif$dm=model.matrix(~ Data_modif$dm - 1, data =
   Data_modif)
Data_modif$cad=model.matrix(~ Data_modif$cad - 1, data =
   Data_modif)
Data_modif$appet=model.matrix(~ Data_modif$appet - 1, data =
   Data_modif)
Data_modif$pe=model.matrix(~ Data_modif$pe - 1, data =
   Data_modif)
Data_modif$ane=model.matrix(~ Data_modif$ane - 1, data =
   Data_modif)
Data_modif$`Data_CKD$Gen` =model.matrix(~
   Data_modif$`Data_CKD$Gen`  - 1, data = Data_modif)

Data_modif=Data_modif[,-15]
Data_modif=Data_modif[,-14]

Data_modif=as.data.frame(Data_modif)
```

```
sg1=Data_modif$sg[,1]
sg2= Data_modif$sg[,2]
sg3=  Data_modif$sg[,3]
sg4= Data_modif$sg[,4]
sg5=Data_modif$sg[,5]
Data_modif=cbind(Data_modif,sg1,sg2,sg3,sg4,sg5)

al1=Data_modif$al[,1]
al2=Data_modif$al[,2]
al3=Data_modif$al[,3]
al4=Data_modif$al[,4]
al5=Data_modif$al[,5]
al6=Data_modif$al[,6]
Data_modif=cbind(Data_modif,al1,al2,al3,al4,al5,al6)

su1=Data_modif$su[,1]
su2=Data_modif$su[,2]
su3=Data_modif$su[,3]
su4=Data_modif$su[,4]
su5=Data_modif$su[,5]
su6=Data_modif$su[,6]
Data_modif=cbind(Data_modif,su1,su2,su3,su4,su5,su6)

rbc1=Data_modif$rbc[,1]
rbc2=Data_modif$rbc[,2]
Data_modif=cbind(Data_modif,rbc1,rbc2)

pc1=Data_modif$pc[,1]
pc2=Data_modif$pc[,2]
Data_modif=cbind(Data_modif,pc1,pc2)

pcc1=Data_modif$pcc[,1]
pcc2=Data_modif$pcc[,2]
Data_modif=cbind(Data_modif,pcc1,pcc2)
```

```
ba1=Data_modif$ba[,1]
ba2=Data_modif$ba[,2]
Data_modif=cbind(Data_modif,ba1,ba2)

htn1=Data_modif$htn[,1]
htn2=Data_modif$htn[,2]
Data_modif=cbind(Data_modif,htn1,htn2)

dm1=Data_modif$dm[,1]
dm2=Data_modif$dm[,2]
Data_modif=cbind(Data_modif,dm1,dm2)

cad1=Data_modif$cad[,1]
cad2=Data_modif$cad[,2]
Data_modif=cbind(Data_modif,cad1,cad2)

appet1=Data_modif$appet[,1]
appet2=Data_modif$appet[,2]
Data_modif=cbind(Data_modif,appet1,appet2)

pe1=Data_modif$pe[,1]
pe2=Data_modif$pe[,2]
Data_modif=cbind(Data_modif,pe1,pe2)

ane1=Data_modif$ane[,1]
ane2=Data_modif$ane[,2]
Data_modif=cbind(Data_modif,ane1,ane2)

Gen1=Data_modif$`Data_CKD$Gen`[,1]
Gen2=Data_modif$`Data_CKD$Gen`[,2]
Data_modif=cbind(Data_modif,Gen1,Gen2)

Data_modif=Data_modif[,-1:-14]
```

División de la base de datos en partición de entrenamiento y prueba

```
cant_datos = 400
set.seed(137)
r = sample(cant_datos)
Data.modif.r = Data_modif[r,]
train = 0.7
test = 1-train

# Metodo Holdout
training.set.0 = Data.modif.r [(1:(cant_datos*train)),]
test.set.0  = Data.modif.r [((cant_datos*train+1):cant_datos),]

training.set.0$etapa=as.factor(training.set.0$etapa)
test.set.0$etapa=as.factor(test.set.0$etapa)
```

Anexo 3: Primer Modelo SVM - Todas las variables

K-fold Cross validation

```
# Secuencia de valores de costo para explorar
cost_values <- 10^(0:3)
accuracy.vector.0 <- numeric(length(cost_values))

tune.result.0 <- tune.svm(etapa ~ .,
                          data = training.set.0,
                          cost = cost_values,
                          kernel = "linear")

for (i in seq_along(cost_values)) {

  accuracy.vector.0[i]=tune.result.0$performances$error[i]
  accuracy.vector.0[i]=1-accuracy.vector.0[i]
}
```

```
# Graficar la curva de precisión en función del costo
plot(cost_values, accuracy.vector.0, type = "b", pch = 16, col
   = "blue",lwd = 3,
      xlab = "Cost", ylab = "Accuracy",
      main = "Curva de Precisión vs Costo")

text(cost_values, accuracy.vector.0, labels =
   round(accuracy.vector.0, 3), pos = 2.8, cex = 1.0, col =
   "blue")
abline(h = accuracy.vector.0, col = "red", lty = 2, lwd = 2)

# Graficar el error en función del costo con mejoras
plot( tune.result.0$performances$error ~
   tune.result.0$performances$cost, pch = 16, col = "blue", lwd
   = 2,
       type = "l", xlab = "Cost", ylab = "Error",
       main = "Curva de Error vs Cost")
text(tune.result.0$performances$error,
   tune.result.0$performances$cost, labels =
   round(accuracy.vector.0, 3), pos = 2.8, cex = 1.4, col =
   "blue")

best.model.index.0 <- tune.result.0$best.model
# Obtener las métricas del mejor modelo
best.model.index.0 <-
   which.min(tune.result.0$performances$error)

best.model.metrics.0 <-
   tune.result.0$performances[best.model.index.0, ]

# Acceder a la precisión del mejor modelo
best.model.accuracy.0 <- 1 - best.model.metrics.0$error
print(paste("Accuracy del mejor modelo:",
   round(best.model.accuracy.0, 3)))
```

Entrenamiento con los mejores hiperparámetros y prueba

```
clasificador.00 <- svm(etapa ~ .,
                       data = training.set.0,
                       type = 'C-classification',
                       kernel = 'linear',
                       cost=tune.result.0$best.parameters$cost)

y.pred.0 <- predict(clasificador.00, newdata =
   training.set.0[,-12])
cm.0 <- table(y.pred.0,training.set.0$etapa)

mc = confusionMatrix(y.pred.0,training.set.0$etapa)
mc1 = t(mc$table) # Matriz de Confusion

matriz_confusion=cm.0

clases <- rownames(matriz_confusion)
#  Clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
accuracies <- numeric(length(clases_evaluables))
# Accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
  accuracies[clase] <- accuracy_clase
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
```

```
    "\n")
}

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad

set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(training.set.0$etapa),  # Las etiquetas
     reales
  pred = as.factor(y.pred.0)  # Las predicciones
)

# Calcular la matriz de confusión
cm <- conf_mat(truth_predicted, obs, pred)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)

autoplot(cm, type = "heatmap") +
  scale_fill_gradient(low = azul_crema_rgb, high =
     "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
```

```
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")

# ----------------------- Prueba

y.pred.00 <- predict(clasificador.00, newdata = test.set.0)
cm.00 <- table(y.pred.00,test.set.0$etapa)

matriz_confusion=cm.00
# Clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
  accuracies[clase] <- accuracy_clase
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}

mc.00 = confusionMatrix(y.pred.00,test.set.0$etapa)
m.c.00 = t(mc.00$table) # Matriz de Confusion
```

```
acc_total.00 = mc.00$overall[1] # Accuracy General
prec.00 = mc.00$byClass[,5] # Precision
F.measure.00 = mc.00$byClass[,7] # F-measure
sensi.00 = mc.00$byClass[,1] # Sensibilidad
especi.00 = mc.00$byClass[,2] # Especificidad

set.seed(123)
truth_predicted.00 <- data.frame(
  obs.00 = as.factor(test.set.0$etapa),  # Las etiquetas reales
  pred.00 = as.factor(y.pred.00)  # Las predicciones
)

# Matriz de confusión
cm.00 <- conf_mat(truth_predicted.00, obs.00, pred.00)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)
autoplot(cm.00, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
```

```
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Prueba",
       subtitle = "Valores Reales")

# ------------------------ Curva ROC Y AUC
library(e1071)

num_classes=6

clasificador.00 <- svm(etapa ~ .,
                       data = training.set.0,
                       type = 'C-classification',
                       kernel = 'linear',
                       cost=tune.result.0$best.parameters$cost,
                       probability = TRUE)
library(pROC)

y_true  <- test.set.0$etapa # Reemplaza test.set.0_t con tus
   etiquetas reales

predicciones_prob <- predict(clasificador.00, test.set.0,
   probability = TRUE)

#y_scores<- attr(predictions, "probabilities")  # Reemplaza
   ann.net y test.set.0 con tus datos reales y predichos

# Transformar las etiquetas reales en un formato binario para
   cada clase
y_true_binary <- lapply(1:6, function(i)
   ifelse(test.set.0$etapa == i, 1, 0))



# Inicializar una lista para almacenar las curvas ROC y los
```

```
   AUC para cada clase
roc_curves <- list()
auc_values <- numeric()

# Calcular la curva ROC y el AUC para cada clase utilizando la
   estrategia OvR
for (i in 0:5) {
  # Convertir la clase actual a binario (1 si es la clase i, 0
     si no)
  y_true <- ifelse(test.set.0$etapa == i, 1, 0)

  # Extraer las probabilidades predichas para la clase i
  y_scores <- attr(predicciones_prob, "probabilities")[, i +
     1]  # Se suma 1 porque R indexa desde 1

  # Calcular la curva ROC
  roc_obj <- roc(y_true, y_scores)

  # Suavizar la curva ROC
  smoothed_roc_obj <- smooth(roc_obj, method = "density")

  # Almacenar la curva ROC suavizada
  roc_curves[[i + 1]] <- smoothed_roc_obj

  # Calcular el AUC
  auc_values[i + 1] <- auc(smoothed_roc_obj)
}


library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")

# Imprimir los valores AUC para cada clase
print(auc_values)
```

```
# Imprimir las curvas ROC para cada clase
plot(roc_curves[[1]], col = "red", main = "Curvas ROC
   Suavizada - Modelo",lwd=1)
for (i in 1:6) {
  plot(roc_curves[[i]], add = TRUE, col = colors[i], lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_values[i],
     3), "\n"), col = colors[i], pos = 1)  # Imprimir el AUC
     para las clases restantes
}

# Agregar leyenda
legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 0.8, lwd = 5, xjust = 1,yjust = 1)
```

Anexo 4: Segundo Modelo SVM - 6 variables

```
#Tomamos esas variantes y creamos unos nuevos datasets

Data.model.1= Data.modif.r

# Metodo Holdout
training.set.1 = Data.model.1 [(1:(cant_datos*train)),]
training.set.1 = cbind(training.set.1[1:12])
training.set.1 = training.set.1[,-2:-3]
training.set.1 = training.set.1[,-4:-5]
training.set.1 =training.set.1[,-6]

test.set.1 = Data.model.1 [((cant_datos*train+1):cant_datos),]
test.set.1 = cbind(test.set.1[1:12])
test.set.1 = test.set.1[,-2:-3]
test.set.1 = test.set.1[,-4:-5]
test.set.1 = test.set.1[,-6]
```

```
# Inicializar vectores para almacenar los resultados
accuracy.vector.1 <- numeric(length(cost_values))

# Realizar tune.svm para encontrar el mejor valor de costo

tune.result.1 <- tune.svm(etapa ~ .,
                          data = training.set.1,
                          cost = cost_values,
                          kernel = "linear")

for (i in seq_along(cost_values)) {

  accuracy.vector.1[i]=tune.result.1$performances$error[i]
  accuracy.vector.1[i]=1-accuracy.vector.1[i]
}

# Graficar la curva de precisión en función del costo
plot(cost_values, accuracy.vector.1, type = "b", pch = 16, col
   = "blue",lwd = 3,
      xlab = "Cost", ylab = "Accuracy",
      main = "Curva de Precisión vs Costo")

text(cost_values, accuracy.vector.1, labels =
   round(accuracy.vector.1, 3), pos = 3, cex = 1.4, col =
   "blue")
abline(h = accuracy.vector.1, col = "red", lty = 2, lwd = 2)

# Graficar el error en función del costo con mejoras
plot( tune.result.1$performances$error ~
   tune.result.1$performances$cost, pch = 16, col = "blue", lwd
   = 2,
       type = "l", xlab = "Cost", ylab = "Error",
       main = "Curva de Error vs Cost")
```

```
best.model.index.1 <- tune.result.1$best.model
# Obtener las métricas del mejor modelo
best.model.index.1 <-
   which.min(tune.result.1$performances$error)

best.model.metrics.1 <-
   tune.result.1$performances[best.model.index.1, ]

# Acceder a la precisión del mejor modelo
best.model.accuracy.1 <- 1 - best.model.metrics.1$error
print(paste("Accuracy del mejor modelo:",
   round(best.model.accuracy.1, 3)))

clasificador.11 <- svm(etapa ~ .,
                       data = training.set.1,
                       type = 'C-classification',
                       kernel = 'linear',
                       cost=tune.result.1$best.parameters$cost)


y.pred.1 <- predict(clasificador.11, newdata =
   training.set.1[,-7])
cm.1 <- table(y.pred.1,training.set.1$etapa)

matriz_confusion=cm.1
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
```

```
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
  accuracies[clase] <- accuracy_clase
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}

mc.1 = confusionMatrix(y.pred.1,training.set.1$etapa)
m.c.1 = t(mc.1$table) # Matriz de Confusion

acc_total.1 = mc.1$overall[1] # Accuracy General
prec.1 = mc.1$byClass[,5] # Precision
F.measure.1 = mc.1$byClass[,7] # F-measure
sensi.1 = mc.1$byClass[,1] # Sensibilidad
especi.1 = mc.1$byClass[,2] # Especificidad

set.seed(123)
truth_predicted.1 <- data.frame(
  obs.1 = as.factor(training.set.1$etapa),  # Las etiquetas
     reales
  pred.1 = as.factor(y.pred.1)  # Las predicciones
)

# Calcular la matriz de confusión
cm.1 <- conf_mat(truth_predicted.1, obs.1, pred.1)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)
# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm.1, type = "heatmap") +
  scale_fill_gradient(low = rgb(207, 221, 233, maxColorValue =
```

```
    255), high = "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
          = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")

# -------------------------- Prueba
y.pred.11 <- predict(clasificador.11, newdata =
   test.set.1[,-7])
cm.11 <- table(y.pred.11,test.set.1$etapa)

matriz_confusion=cm.11
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
```

```
    != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
    clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase

  # Imprimir el resultado para la clase actual
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
    "\n")
}

mc.11 = confusionMatrix(y.pred.11,test.set.1$etapa)
m.c.11 = t(mc.11$table) # Matriz de Confusion

acc_total.11 = mc.11$overall[1] # Accuracy General
prec.11 = mc.11$byClass[,5] # Precision
F.measure.11 = mc.11$byClass[,7] # F-measure
sensi.11 = mc.11$byClass[,1] # Sensibilidad
especi.11 = mc.11$byClass[,2] # Especificidad

set.seed(123)
truth_predicted.11 <- data.frame(
  obs.11 = as.factor(test.set.1$etapa),  # Las etiquetas reales
  pred.11 = as.factor(y.pred.11)  # Las predicciones
)

# Calcular la matriz de confusión
cm.11 <- conf_mat(truth_predicted.11, obs.11, pred.11)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)
```

```
# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm.11, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Prueba",
       subtitle = "Valores Reales")

# ----------------------- Curva ROC Y AUC
library(e1071)

clasificador.11 <- svm(etapa ~ .,
                       data = training.set.1,
                       type = 'C-classification',
                       kernel = 'linear',
                       cost=tune.result.1$best.parameters$cost,
                       probability = TRUE)

library(pROC)
```

```
y_true  <- test.set.1$etapa # Reemplaza test.set.0_t con tus
   etiquetas reales

predicciones_prob <- predict(clasificador.11, test.set.1,
   probability = TRUE)



# Transformar las etiquetas reales en un formato binario para
   cada clase
y_true_binary <- lapply(1:6, function(i)
   ifelse(test.set.1$etapa == i, 1, 0))



# Inicializar una lista para almacenar las curvas ROC y los
   AUC para cada clase
roc_curves <- list()
auc_values <- numeric()

# Calcular la curva ROC y el AUC para cada clase utilizando la
   estrategia OvR
for (i in 0:5) {
  # Convertir la clase actual a binario (1 si es la clase i, 0
     si no)
  y_true <- ifelse(test.set.1$etapa == i, 1, 0)

  # Extraer las probabilidades predichas para la clase i
  y_scores <- attr(predicciones_prob, "probabilities")[, i +
     1]  # Se suma 1 porque R indexa desde 1

  # Calcular la curva ROC
  roc_obj <- roc(y_true, y_scores)
```

```
  # Suavizar la curva ROC
  smoothed_roc_obj <- smooth(roc_obj, method = "density")

  # Almacenar la curva ROC suavizada
  roc_curves[[i + 1]] <- smoothed_roc_obj

  # Calcular el AUC
  auc_values[i + 1] <- auc(smoothed_roc_obj)
}


library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")

# Imprimir los valores AUC para cada clase
print(auc_values)

# Imprimir las curvas ROC para cada clase
plot(roc_curves[[1]], col = "red", main = "Curvas ROC
   Suavizada - Modelo",lwd=1)
for (i in 1:6) {
  plot(roc_curves[[i]], add = TRUE, col = colors[i], lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_values[i],
     3), "\n"), col = colors[i], pos = 1)  # Imprimir el AUC
     para las clases restantes
}

# Agregar leyenda
legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 0.8, lwd = 5, xjust = 1,yjust = 1)
```

Anexo 5: Tercer Modelo SVM - 2 variables

```
training.set.2 = training.set.1
training.set.2 = training.set.2[,-1:-2]
training.set.2 = training.set.2[,-2]
training.set.2 = training.set.2[,-3]

test.set.2 = test.set.1
test.set.2 = test.set.2[,-1:-2]
test.set.2 = test.set.2[,-2]
test.set.2 = test.set.2[,-3]

# ------ Implementacion de k-fold Cross Validation
# Inicializar vectores para almacenar los resultados
accuracy.vector.2 <- numeric(length(cost_values))

tune.result.2 <- tune.svm(etapa ~ .,
                          data = training.set.2,
                          cost = cost_values,
                          kernel = "radial")

for (i in seq_along(cost_values)) {

  accuracy.vector.2[i]=tune.result.2$performances$error[i]
  accuracy.vector.2[i]=1-accuracy.vector.2[i]
}

# Graficar la curva de precisión en función del costo
plot(cost_values, accuracy.vector.2, type = "b", pch = 16, col
   = "blue",lwd = 3,
      xlab = "Cost", ylab = "Accuracy",
      main = "Curva de Precisión vs Costo")

# Agregar etiquetas de precisión a los puntos
text(cost_values, accuracy.vector.2, labels =
   round(accuracy.vector.2, 3), pos = 3, cex = 1.4, col =
```

```
   "blue")

# Agregar líneas de referencia
abline(h = accuracy.vector.2, col = "red", lty = 2, lwd = 2)
   # Línea para la máxima precisión
# Graficar el error en función del costo con mejoras
plot( tune.result.2$performances$error ~
   tune.result.2$performances$cost, pch = 16, col = "blue", lwd
   = 2,
       type = "l", xlab = "Cost", ylab = "Error",
       main = "Curva de Error vs Cost")

best.model.index.2 <- tune.result.2$best.model
# Obtener las métricas del mejor modelo
best.model.index.2 <-
   which.min(tune.result.2$performances$error)

best.model.metrics.2 <-
   tune.result.2$performances[best.model.index.2, ]

# Acceder a la precisión del mejor modelo
best.model.accuracy.2 <- 1 - best.model.metrics.2$error
print(paste("Accuracy del mejor modelo:",
   round(best.model.accuracy.2, 3)))

# -------------------------  Entrenamiento

clasificador.22 <- svm(etapa ~ .,
                       data = training.set.2,
                       type = 'C-classification',
                       kernel = 'radial',
                       cost=tune.result.2$best.parameters$cost)
```

```
y.pred.2 <- predict(clasificador.22, newdata =
   training.set.2[,-3])
cm.2 <- table(y.pred.2,training.set.2$etapa)
matriz_confusion=cm.2
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
  accuracies[clase] <- accuracy_clase
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}

mc.2 = confusionMatrix(y.pred.2,training.set.2$etapa)
m.c.2 = t(mc.2$table) # Matriz de Confusion


acc_total.2 = mc.2$overall[1] # Accuracy General
prec.2 = mc.2$byClass[,5] # Precision
F.measure.2 = mc.2$byClass[,7] # F-measure
sensi.2 = mc.2$byClass[,1] # Sensibilidad
especi.2 = mc.2$byClass[,2] # Especificidad
```

```r
set.seed(123)
truth_predicted.2 <- data.frame(
  obs.2 = as.factor(training.set.2$etapa),  # Las etiquetas
     reales
  pred.2 = as.factor(y.pred.2)  # Las predicciones
)

# Calcular la matriz de confusión
cm.2 <- conf_mat(truth_predicted.2, obs.2, pred.2)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)
# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm.2, type = "heatmap") +
  scale_fill_gradient(low = azul_crema_rgb , high =
     "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")
```

```
# ------------------- Prueba


y.pred.22 <- predict(clasificador.22, newdata =
   test.set.2[,-3])
cm.22 <- table(y.pred.22,test.set.2$etapa)

matriz_confusion=cm.22
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
  accuracies[clase] <- accuracy_clase
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}


mc.22 = confusionMatrix(y.pred.22,test.set.2$etapa)
m.c.22 = t(mc.22$table) # Matriz de Confusion

acc_total.22 = mc.22$overall[1] # Accuracy General
prec.22 = mc.22$byClass[,5] # Precision
```

```
F.measure.22 = mc.22$byClass[,7] # F-measure
sensi.22 = mc.22$byClass[,1] # Sensibilidad
especi.22 = mc.22$byClass[,2] # Especificidad

set.seed(123)
truth_predicted.22 <- data.frame(
  obs.22 = as.factor(test.set.2$etapa),  # Las etiquetas reales
  pred.22 = as.factor(y.pred.22)  # Las predicciones
)

# Calcular la matriz de confusión
cm.22 <- conf_mat(truth_predicted.22, obs.22, pred.22)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)
# Graficar la matriz de confusión
autoplot(cm.22, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz␣de␣Confusión␣-␣Prueba",
```

```
        subtitle = "Valores Reales")

# ------------------------ Curva ROC Y AUC
library(e1071)

# Entrenar el modelo SVM

clasificador.00 <- svm(etapa ~ .,
                       data = training.set.0,
                       cost=tune.result.0$best.parameters$cost,
                       probability = TRUE)

clasificador.22 <- svm(etapa ~ .,
                       data = training.set.2,
                       type = 'C-classification',
                       kernel = 'radial',
                       cost=tune.result.2$best.parameters$cost,
                       probability = TRUE)

library(pROC)

y_true  <- test.set.2$etapa # Reemplaza test.set.0_t con tus
   etiquetas reales

predicciones_prob <- predict(clasificador.22, test.set.2,
   probability = TRUE)


# Transformar las etiquetas reales en un formato binario para
   cada clase
y_true_binary <- lapply(1:6, function(i)
   ifelse(test.set.2$etapa == i, 1, 0))
```

```r
# Inicializar una lista para almacenar las curvas ROC y los
   AUC para cada clase
roc_curves <- list()
auc_values <- numeric()

# Calcular la curva ROC y el AUC para cada clase utilizando la
   estrategia OvR
for (i in 0:5) {
  # Convertir la clase actual a binario (1 si es la clase i, 0
     si no)
  y_true <- ifelse(test.set.0$etapa == i, 1, 0)

  # Extraer las probabilidades predichas para la clase i
  y_scores <- attr(predicciones_prob, "probabilities")[, i +
     1]  # Se suma 1 porque R indexa desde 1

  # Calcular la curva ROC
  roc_obj <- roc(y_true, y_scores)

  # Suavizar la curva ROC
  smoothed_roc_obj <- smooth(roc_obj, method = "density")

  # Almacenar la curva ROC suavizada
  roc_curves[[i + 1]] <- smoothed_roc_obj

  # Calcular el AUC
  auc_values[i + 1] <- auc(smoothed_roc_obj)
}

library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")

# Imprimir los valores AUC para cada clase
```

```
print(auc_values)

# Imprimir las curvas ROC para cada clase
plot(roc_curves[[1]], col = "red", main = "Curvas ROC
   Suavizada - Modelo",lwd=1)
for (i in 1:6) {
  plot(roc_curves[[i]], add = TRUE, col = colors[i], lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_values[i],
     3), "\n"), col = colors[i], pos = 1)  # Imprimir el AUC
     para las clases restantes
}

# Agregar leyenda
legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 0.8, lwd = 5, xjust = 1,yjust = 1)
```

Anexo 6: Cuarto Modelo SVM - 1 variable

```
# Metodo Holdout

training.set.3  = training.set.2
training.set.3  = training.set.3[,-2]

test.set.3 = test.set.2
test.set.3 = test.set.3[,-2]

# ------Implementacion de k-fold Cross Validation
accuracy.vector.3 <- numeric(length(cost_values))

tune.result.3 <- tune.svm(etapa ~ .,
                          data = training.set.3,
                          cost = cost_values,
                          kernel = "radial")
```

```
for (i in seq_along(cost_values)) {

  accuracy.vector.3[i]=tune.result.3$performances$error[i]
  accuracy.vector.3[i]=1-accuracy.vector.3[i]
}

# Graficar la curva de precisión en función del costo
plot(cost_values, accuracy.vector.3, type = "b", pch = 16, col
   = "blue",lwd = 3,
      xlab = "Cost", ylab = "Accuracy",
      main = "Curva de Precisión vs Costo")

text(cost_values, accuracy.vector.3, labels =
   round(accuracy.vector.3, 3), pos = 3, cex = 1.4, col =
   "blue")
abline(h = accuracy.vector.3, col = "red", lty = 2, lwd = 2)

# Graficar el error en función del costo con mejoras
plot( tune.result.3$performances$error ~
   tune.result.3$performances$cost, pch = 16, col = "blue", lwd
   = 2,
       type = "l", xlab = "Cost", ylab = "Error",
       main = "Curva de Error vs Cost")

best.model.index.3 <- tune.result.3$best.model
# Obtener las métricas del mejor modelo
best.model.index.3 <-
   which.min(tune.result.3$performances$error)

best.model.metrics.3 <-
   tune.result.3$performances[best.model.index.3, ]

# Acceder a la precisión del mejor modelo
best.model.accuracy.3 <- 1 - best.model.metrics.3$error
```

```
print(paste("Accuracy del mejor modelo:",
   round(best.model.accuracy.3, 3)))

# ------------------------------- Entrenamiento
clasificador.33 <- svm(etapa ~ sc,
                       data = training.set.3,
                       type = 'C-classification',
                       kernel = 'radial',
                       cost= 1)

y.pred.3 <- predict(clasificador.33, newdata = training.set.3)
cm.3 <- table(y.pred.3,training.set.3$etapa)
mc.3 = confusionMatrix(y.pred.3,training.set.3$etapa)
m.c.3 = t(mc.3$table) # Matriz de Confusion
matriz_confusion=cm.3

# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]

# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase
```

```
  # Imprimir el resultado para la clase actual
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}

acc_total.3 = mc.3$overall[1] # Accuracy General
prec.3 = mc.3$byClass[,5] # Precision
F.measure.3 = mc.3$byClass[,7] # F-measure
sensi.3 = mc.3$byClass[,1] # Sensibilidad
especi.3 = mc.3$byClass[,2] # Especificidad

set.seed(123)
truth_predicted.3 <- data.frame(
  obs.3 = as.factor(training.set.3$etapa),  # Las etiquetas
     reales
  pred.3 = as.factor(y.pred.3)  # Las predicciones
)

# Calcular la matriz de confusión
cm.3 <- conf_mat(truth_predicted.3, obs.3, pred.3)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)
# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm.3, type = "heatmap") +
  scale_fill_gradient(low = azul_crema_rgb, high =
     "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
```

```
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")

# ------------------------------ Prueba
y.pred.33 <- predict(clasificador.33, newdata = test.set.3)
cm.33 <- table(y.pred.33,test.set.3$etapa)

matriz_confusion=cm.33
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase
  # Imprimir el resultado para la clase actual
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}
```

```
mc.33 = confusionMatrix(y.pred.33,test.set.3$etapa)
m.c.33 = t(mc.33$table) # Matriz de Confusion

acc_total.33 = mc.33$overall[1] # Accuracy General
prec.33 = mc.33$byClass[,5] # Precision
F.measure.33 = mc.33$byClass[,7] # F-measure
sensi.33 = mc.33$byClass[,1] # Sensibilidad
especi.33 = mc.33$byClass[,2] # Especificidad

set.seed(123)
truth_predicted.33 <- data.frame(
  obs.33 = as.factor(test.set.3$etapa),  # Las etiquetas reales
  pred.33 = as.factor(y.pred.33)  # Las predicciones
)

# Calcular la matriz de confusión
cm.33 <- conf_mat(truth_predicted.33, obs.33, pred.33)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)
# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm.33, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
```

```
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Prueba",
       subtitle = "Valores Reales")

clasificador.33 <- svm(etapa ~ sc,
                       data = training.set.3,
                       type = 'C-classification',
                       kernel = 'radial',
                       cost= 1,
                       probability = TRUE)


library(pROC)

y_true  <- test.set.3$etapa # Reemplaza test.set.0_t con tus
   etiquetas reales

predicciones_prob <- predict(clasificador.33, test.set.3,
   probability = TRUE)

y_scores<- attr(predictions, "probabilities")  # Reemplaza
   ann.net y test.set.0 con tus datos reales y predichos

# Transformar las etiquetas reales en un formato binario para
   cada clase
y_true_binary <- lapply(1:6, function(i)
   ifelse(test.set.3$etapa == i, 1, 0))
```

```r
# Inicializar una lista para almacenar las curvas ROC y los
   AUC para cada clase
roc_curves <- list()
auc_values <- numeric()

# Calcular la curva ROC y el AUC para cada clase utilizando la
   estrategia OvR
for (i in 0:5) {
  # Convertir la clase actual a binario (1 si es la clase i, 0
     si no)
  y_true <- ifelse(test.set.3$etapa == i, 1, 0)


  # Calcular la curva ROC
  roc_obj <- roc(y_true, y_scores)

  # Suavizar la curva ROC
  smoothed_roc_obj <- smooth(roc_obj, method = "density")

  # Almacenar la curva ROC suavizada
  roc_curves[[i + 1]] <- smoothed_roc_obj

  # Calcular el AUC
  auc_values[i + 1] <- auc(smoothed_roc_obj)
}

library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")

# Imprimir los valores AUC para cada clase
print(auc_values)

# Imprimir las curvas ROC para cada clase
plot(roc_curves[[1]], col = "red", main = "Curvas␣ROC␣
```

```
   Suavizada␣-␣Modelo",lwd=1)
for (i in 1:6) {
  plot(roc_curves[[i]], add = TRUE, col = colors[i], lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_values[i],
     3), "\n"), col = colors[i], pos = 1)  # Imprimir el AUC
     para las clases restantes
}

# Agregar leyenda
legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 0.8, lwd = 5, xjust = 1,yjust = 1)
```

Anexo 7: Código modelos ANN

```
library(readr)
library(corrplot)
library(caTools)
library(neuralnet)
library(NeuralNetTools)
library(caret)
library(e1071)
library(neuralnet)
library(caret)
library(yardstick)
library(ggplot2)


Data_modif <- read_csv("C:/Users/alext/Desktop/P_Tesis/Base␣de␣
   datos/Data_modif.csv")
#View(Data_modif)

Data_modif$sg=as.factor(Data_modif$sg)
Data_modif$al=as.factor(Data_modif$al)
```

```
Data_modif$su=as.factor(Data_modif$su)
Data_modif$rbc=as.factor(Data_modif$rbc)
Data_modif$pc=as.factor(Data_modif$pc)
Data_modif$pcc=as.factor(Data_modif$pcc)
Data_modif$ba=as.factor(Data_modif$ba)
Data_modif$htn=as.factor(Data_modif$htn)
Data_modif$dm=as.factor(Data_modif$dm)
Data_modif$cad=as.factor(Data_modif$cad)
Data_modif$appet=as.factor(Data_modif$appet)
Data_modif$pe=as.factor(Data_modif$pe)
Data_modif$ane=as.factor(Data_modif$ane)
Data_modif$classification=as.factor(Data_modif$classification)
Data_modif$`Data_CKD$Gen`=as.factor(Data_modif$`Data_CKD$Gen`)
Data_modif$etapa=as.factor(Data_modif$etapa)

cuantitative_ckd=cbind(Data_modif[,15],Data_modif[,17:27])

New_prueba=cuantitative_ckd[,-1]
New_prueba=as.data.frame(cbind(New_prueba,Data_modif[,28]))

mat_cor=cor(cuantitative_ckd)
var.1=as.data.frame(cbind(cuantitative_ckd$FGe,cuantitative_ckd$age,
                          cuantitative_ckd$bu,cuantitative_ckd$sc,
                          cuantitative_ckd$hemo,cuantitative_ckd$pcv
                          ,cuantitative_ckd$rc))

nuevos_nombres <- c("FGe", "age", "bu", "sc", "hemo", "pcv",
   "rc")
colnames(var.1) <- nuevos_nombres
mat_cor.1=cor(var.1)

corrplot(mat_cor.1)

pairs(mat_cor.1,upper.panel = NULL,col="blue",lower.panel =
```

```
   panel.smooth)


corrplot(mat_cor)
corrplot(mat_cor, method = "circle", type = "lower", order =
   "original",tl.cex = 1.8, cl.cex = 1.8)
pairs(mat_cor,upper.panel = NULL,col="blue",lower.panel =
   panel.smooth)

#Cuali One - Hot - Encoding
Data_modif$sg=model.matrix(~ Data_modif$sg - 1, data =
   Data_modif)
Data_modif$al=model.matrix(~ Data_modif$al - 1, data =
   Data_modif)
Data_modif$su=model.matrix(~ Data_modif$su - 1, data =
   Data_modif)
Data_modif$rbc=model.matrix(~ Data_modif$rbc - 1, data =
   Data_modif)
Data_modif$pc=model.matrix(~ Data_modif$pc - 1, data =
   Data_modif)
Data_modif$pcc=model.matrix(~ Data_modif$pcc - 1, data =
   Data_modif)
Data_modif$ba=model.matrix(~ Data_modif$ba - 1, data =
   Data_modif)
Data_modif$htn=model.matrix(~ Data_modif$htn - 1, data =
   Data_modif)
Data_modif$dm=model.matrix(~ Data_modif$dm - 1, data =
   Data_modif)
Data_modif$cad=model.matrix(~ Data_modif$cad - 1, data =
   Data_modif)
Data_modif$appet=model.matrix(~ Data_modif$appet - 1, data =
   Data_modif)
Data_modif$pe=model.matrix(~ Data_modif$pe - 1, data =
   Data_modif)
```

```
Data_modif$ane=model.matrix(~ Data_modif$ane - 1, data =
   Data_modif)
Data_modif$`Data_CKD$Gen` =model.matrix(~
   Data_modif$`Data_CKD$Gen`  - 1, data = Data_modif)

#Cuanti Escalado
Data_modif$FGe=scale(Data_modif$FGe)
Data_modif$age=scale(Data_modif$age)
Data_modif$bp=scale(Data_modif$bp)
Data_modif$bgr=scale(Data_modif$bgr)
Data_modif$bu=scale(Data_modif$bu)
Data_modif$sc=scale(Data_modif$sc)
Data_modif$sod=scale(Data_modif$sod)
Data_modif$pot=scale(Data_modif$pot)
Data_modif$hemo=scale(Data_modif$hemo)
Data_modif$pcv=scale(Data_modif$pcv)
Data_modif$wc=scale(Data_modif$wc)
Data_modif$rc=scale(Data_modif$rc)

Data_modif=Data_modif[,-15]
Data_modif=Data_modif[,-14]

Data_modif=as.data.frame(Data_modif)

sg1=Data_modif$sg[,1]
sg2= Data_modif$sg[,2]
sg3=  Data_modif$sg[,3]
sg4= Data_modif$sg[,4]
sg5=Data_modif$sg[,5]
Data_modif=cbind(Data_modif,sg1,sg2,sg3,sg4,sg5)

al1=Data_modif$al[,1]
al2=Data_modif$al[,2]
al3=Data_modif$al[,3]
```

```
al4=Data_modif$al[,4]
al5=Data_modif$al[,5]
al6=Data_modif$al[,6]
Data_modif=cbind(Data_modif,al1,al2,al3,al4,al5,al6)

su1=Data_modif$su[,1]
su2=Data_modif$su[,2]
su3=Data_modif$su[,3]
su4=Data_modif$su[,4]
su5=Data_modif$su[,5]
su6=Data_modif$su[,6]
Data_modif=cbind(Data_modif,su1,su2,su3,su4,su5,su6)


rbc1=Data_modif$rbc[,1]
rbc2=Data_modif$rbc[,2]
Data_modif=cbind(Data_modif,rbc1,rbc2)

pc1=Data_modif$pc[,1]
pc2=Data_modif$pc[,2]
Data_modif=cbind(Data_modif,pc1,pc2)

pcc1=Data_modif$pcc[,1]
pcc2=Data_modif$pcc[,2]
Data_modif=cbind(Data_modif,pcc1,pcc2)

ba1=Data_modif$ba[,1]
ba2=Data_modif$ba[,2]
Data_modif=cbind(Data_modif,ba1,ba2)

htn1=Data_modif$htn[,1]
htn2=Data_modif$htn[,2]
Data_modif=cbind(Data_modif,htn1,htn2)
```

```
dm1=Data_modif$dm[,1]
dm2=Data_modif$dm[,2]
Data_modif=cbind(Data_modif,dm1,dm2)

cad1=Data_modif$cad[,1]
cad2=Data_modif$cad[,2]
Data_modif=cbind(Data_modif,cad1,cad2)

appet1=Data_modif$appet[,1]
appet2=Data_modif$appet[,2]
Data_modif=cbind(Data_modif,appet1,appet2)

pe1=Data_modif$pe[,1]
pe2=Data_modif$pe[,2]
Data_modif=cbind(Data_modif,pe1,pe2)

ane1=Data_modif$ane[,1]
ane2=Data_modif$ane[,2]
Data_modif=cbind(Data_modif,ane1,ane2)

Gen1=Data_modif$`Data_CKD$Gen`[,1]
Gen2=Data_modif$`Data_CKD$Gen`[,2]
Data_modif=cbind(Data_modif,Gen1,Gen2)

Data_modif=Data_modif[,-1:-14]

Data_modif$etapa=as.numeric(Data_modif$etapa)

}
```

Anexo 8: Primer modelo ANN - Todas las variables

```
cant_datos = 400
set.seed(137)
```

```
r = sample(cant_datos)
Data.modif.r =Data_modif[r,]
train = 0.7
test = 1-train

# Metodo Holdout
training.set.0 = Data.modif.r [(1:(cant_datos*train)),]
test.set.0  = Data.modif.r [((cant_datos*train+1):cant_datos),]

training.set.0$etapa=as.factor(training.set.0$etapa)
test.set.0$etapa=as.factor(test.set.0$etapa)

# ------ Implementacion de k-fold Cross Validation

# ------------------- ANN
# ------------------ Modelo con todas las variables de Red
   Neuronal

num_folds <- 10

folds <- createFolds(training.set.0$etapa, k = num_folds)
# Inicializar un vector para almacenar los resultados de cada
   fold
accuracy <- numeric(num_folds)
resultados<- numeric(num_folds)

# ------------------------------------------------- ----
   Ten-Fold Cross Validation
# Iterar sobre los folds

for (i in 1:num_folds) {
# Dividir el conjunto de datos en entrenamiento y prueba según
   el fold actual
train_indices <- unlist(folds[-i])
```

```
test_indices <- folds[[i]]

training_set <- training.set.0[train_indices, ]
testing_set <- training.set.0[test_indices, ]

                act.fct = "logistic")


ann.net <- neuralnet(formula = as.factor(etapa) ~ .,
                     data = training_set,
                   hidden = c(10,20,20),
                   stepmax = 1e+05,
                  threshold = 0.01,
                   linear.output = TRUE,
                act.fct = "logistic")




# Realizar predicciones en el conjunto de prueba
predictions <- predict(ann.net, newdata = testing_set, type =
   "response")
predictions = max.col(predictions)
# Calcular la precisión y almacenarla en el vector de
   resultados
accuracy[i] <- sum(predictions == testing_set$etapa) /
   length(testing_set$etapa)
resultados[i]=ann.net$result.matrix[3]
#print(head(ann.net$result.matrix))
}

# Calcular la precisión media de todos los folds
mean_accuracy <- mean(accuracy)
```

```r
# Imprimir la precisión media
print(paste("Mean Accuracy:", mean_accuracy))

# head(ann.net$result.matrix)

#plotnet(ann.net)

# Dividir los datos en conjuntos de entrenamiento y validación
n <- nrow(training.set.0)
index <- sample(1:n, size = round(0.7 * n))  # 70% para
   entrenamiento, 30% para validación
train_data <- training.set.0[index, ]
val_data <- training.set.0[-index, ]

# Monitoreo del rendimiento en el conjunto de validación
   durante el entrenamiento
epochs <- 100
best_accuracy <- 0
best_epoch <- 0


# Función para calcular la precisión en el conjunto de
   validación (clasificación multiclase)
calculate_accuracy<- function(nn_model, validation_data) {
  validation_data_s=validation_data[, 1:11]
  validation_data_s=cbind(validation_data_s,validation_data[,
     13:51])
  predictions <- predict(nn_model, validation_data_s)
  #predictions = max.col(predictions)
  #predicted_classes <- apply(predictions, 1, which.max) - 1
     # Restamos 1 para obtener clases numeradas de 0 a 5
  predicted_classes = max.col(predictions)
  accuracy <- mean(predicted_classes == validation_data$etapa)
  return(accuracy)
```

```
}

for (epoch in 1:epochs) {
  nn_epoch <- neuralnet(formula = etapa ~ .,
                        data = train_data,
                        hidden = c(10,20,20),
                        threshold = 0.01,
                        linear.output = TRUE,
                        act.fct = "logistic"
  )


  accuracy <- calculate_accuracy(nn_epoch, val_data)
  cat("Epoch:", epoch, ", Validation Accuracy:", accuracy,
     "\n")

  # Guardar el modelo con mejor rendimiento en el conjunto de
     validación
  if (accuracy > best_accuracy) {
    best_accuracy <- accuracy
    best_epoch <- epoch
    best_nn <- nn_epoch
  } else {
    # Detener el entrenamiento si la precisión en el conjunto
       de validación comienza a disminuir
    if (epoch - best_epoch > 5) {  # Si la precisión no ha
       mejorado en las últimas 5 épocas
      cat("Early stopping at epoch:", epoch, "\n")
      break
    }
  }
}
```

```
#
   ------------------------------------------------------------------
   Entrenamiento

# Iniciar el temporizador
start_time <- Sys.time()

ann.net = neuralnet(formula = as.factor(etapa) ~ .,
                    data = training.set.0,
                    hidden = c(10,20,20),
                    threshold = 0.01,
                    linear.output = TRUE,
                    act.fct = "logistic",
                    rep=4)


# Detener el temporizador y mostrar el tiempo transcurrido
end_time <- Sys.time()
elapsed_time <- end_time - start_time
cat("Tiempo de entrenamiento:", as.numeric(elapsed_time),
   "segundos\n")

#plot(ann.net, rep = "best")
#plotnet(ann.net)
head(ann.net$result.matrix)

data_pred = predict(ann.net, training.set.0, type = "response")
data_pred = max.col(data_pred)
table(data_pred)
table(training.set.0$etapa)

# Realizar el reemplazo según tus criterios
data_pred_t <- ifelse(data_pred == 1, 0,
                      ifelse(data_pred == 2, 1,
```

```
                                ifelse(data_pred == 3, 2,
                                   ifelse(data_pred == 4, 3,
                                          ifelse(data_pred == 5,
                                             4,
                                          ifelse(data_pred == 6,
                                             5, data_pred))))))

# Realizar el reemplazo según tus criterios
training.set.0_t <- ifelse(training.set.0$etapa == 1, 0,
                           ifelse(training.set.0$etapa == 2, 1,
                                  ifelse(training.set.0$etapa
                                     == 3, 2,
                                    ifelse(training.set.0$etapa
                                       == 4, 3,
                                    ifelse(training.set.0$etapa
                                       == 5, 4,
                                    ifelse(training.set.0$etapa==
                                       6, 5,
                                       training.set.0$etapa))))))

# Realizar el reemplazo según tus criterios
test.set.0_t <- ifelse(test.set.0$etapa == 1, 0,
                       ifelse(test.set.0$etapa == 2, 1,
                              ifelse(test.set.0$etapa == 3, 2,
                                ifelse(test.set.0$etapa == 4,
                                   3,
                                    ifelse(test.set.0$etapa ==
                                       5, 4,
                                    ifelse(test.set.0$etapa==
                                       6, 5,
                                       training.set.0$etapa))))))
cm.0 <- table(data_pred_t ,training.set.0_t)

matriz_confusion=cm.0
```

```
# Obtener nombres de clases
clases <- rownames(matriz_confusion)
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Inicializar vector para almacenar accuracies por clase
accuracies <- numeric(length(clases_evaluables))
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase

  # Imprimir el resultado para la clase actual
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}


mc =
   confusionMatrix(as.factor(data_pred_t),as.factor(training.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
```

```
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad



# Supongamos que tienes las predicciones y las etiquetas
   reales en las variables y.pred.22 y test.set.2$etapa
set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(training.set.0_t),  # Las etiquetas reales
  pred = as.factor(data_pred_t )  # Las predicciones
)

# Calcular la matriz de confusión
cm <- conf_mat(truth_predicted, obs, pred)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)

# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm, type = "heatmap") +
  scale_fill_gradient(low = azul_crema_rgb, high =
     "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
```

```
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")

# ------ Prueba
data_pred.00 = predict(ann.net, test.set.0, type = "response")
data_pred.00 = max.col(data_pred.00)
table(data_pred.00)
table(test.set.0$etapa)

# Realizar el reemplazo según tus criterios
data_pred_t <- ifelse(data_pred.00  == 1, 0,
                      ifelse(data_pred.00  == 2, 1,
                             ifelse(data_pred.00  == 3, 2,
                                    ifelse(data_pred.00  == 4,
                                       3,
                                         ifelse(data_pred.00
                                            == 5, 4,
                                                ifelse(data_pred.00
                                                   == 6,5,
                                                   data_pred.00
                                                   ))))))

cm.00 <- table(data_pred_t,test.set.0_t)

matriz_confusion=cm.00
# Obtener nombres de clases
clases <- rownames(matriz_confusion)
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
```

```
# Inicializar vector para almacenar accuracies por clase
accuracies <- numeric(length(clases_evaluables))
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase

  # Imprimir el resultado para la clase actual
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}


mc =
   confusionMatrix(as.factor(data_pred_t),as.factor(test.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad
```

```r
# Supongamos que tienes las predicciones y las etiquetas
   reales en las variables y.pred.22 y test.set.2$etapa
set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(test.set.0_t),  # Las etiquetas reales
  pred = as.factor(data_pred_t )  # Las predicciones
)

# Calcular la matriz de confusión
cm.00 <- conf_mat(truth_predicted, obs, pred)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)

# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm.00, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
```

```
  labs(title = "Matriz de Confusión - Prueba",
       subtitle = "Valores Reales")

# ------------------------------------ Curva ROC Y AUC


# Instalar y cargar la librería pROC si aún no está instalada
if (!requireNamespace("pROC", quietly = TRUE)) {
  install.packages("pROC")
}
library(pROC)

# Ejemplo de etiquetas verdaderas y puntajes de predicción
   para un problema multiclase
num_classes <- 6

# Generar datos de ejemplo
# Obtener las etiquetas reales y las probabilidades predichas
   para cada clase
y_true  <- test.set.0$etapa  # Reemplaza test.set.0_t con tus
   etiquetas reales
y_scores<- predict(ann.net, test.set.0, type = "response")  #
   Reemplaza ann.net y test.set.0 con tus datos reales y
   predichos


# Construir la curva ROC para cada clase y suavizarla
smoothed_roc_curves <- lapply(1:num_classes,
   function(class_index) {
  # Convertir a binario (uno-vs-rest)
  y_true_binary <- ifelse(y_true == class_index, 1, 0)
  # Calcular la curva ROC para esta clase
  roc_obj <- roc(y_true_binary, y_scores[, class_index])
  # Suavizar la curva ROC
```

```
  smooth(roc_obj, method = "binormal")
})

# Función para obtener el AUC de cada curva ROC
auc_class <- sapply(smoothed_roc_curves, function(roc_obj)
   auc(roc_obj))

# Imprimir el AUC para cada clase
print("AUC para cada clase:")
print(auc_class)

# Colores para las curvas ROC
colors <- rainbow(num_classes)

library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")

# Dibujar las curvas ROC suavizadas para cada clase con
   colores diferentes
plot(smoothed_roc_curves[[1]], col = colors[1], main = "Curvas
   ROC Suavizada - Modelo 1",lwd=5)
for (i in 1:num_classes) {
  plot(smoothed_roc_curves[[i]], add = TRUE, col = colors[i],
     lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_class[i],
     3), "\n"), col = colors[i], pos = 1)  # Imprimir el AUC
     para las clases restantes
}

# Agregar leyenda
legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 1, lwd = 5, xjust = 1,yjust = 1)
```

Anexo 9: Segundo modelo ANN - 6 variables

```
cant_datos = 400
set.seed(137)
r = sample(cant_datos)
Data.modif.r =Data_modif[r,]
train = 0.7
test = 1-train

# Metodo Holdout
training.set.0 = Data.modif.r [(1:(cant_datos*train)),]
training.set.0 = cbind(training.set.0[1:12])
training.set.0 = training.set.0[,-2:-3]
training.set.0 = training.set.0[,-4:-5]
training.set.0 =training.set.0[,-6]



test.set.0  = Data.modif.r [((cant_datos*train+1):cant_datos),]
test.set.0 = cbind(test.set.0[1:12])
test.set.0 = test.set.0[,-2:-3]
test.set.0 = test.set.0[,-4:-5]
test.set.0 = test.set.0[,-6]

training.set.0$etapa=as.factor(training.set.0$etapa)
test.set.0$etapa=as.factor(test.set.0$etapa)

# Iterar sobre los folds
for (i in 1:num_folds) {
# Dividir el conjunto de datos en entrenamiento y prueba según
   el fold actual
train_indices <- unlist(folds[-i])
test_indices <- folds[[i]]

training_set <- training.set.0[train_indices, ]
testing_set <- training.set.0[test_indices, ]
```

```
# 0.74
ann.net <- neuralnet(formula = as.factor(etapa) ~ .,
                     data = training_set,
                    hidden = c(5),
                    stepmax = 1e+05,
                  threshold = 0.01,
                    linear.output = TRUE,
                  act.fct = "logistic")


}

# Calcular la precisión media de todos los folds
mean_accuracy <- mean(accuracy)

# Imprimir la precisión media
print(paste("Mean Accuracy:", mean_accuracy))

# head(ann.net$result.matrix)

#plotnet(ann.net)

# Dividir los datos en conjuntos de entrenamiento y validación
n <- nrow(training.set.0)
index <- sample(1:n, size = round(0.7 * n))  # 70% para
   entrenamiento, 30% para validación
train_data <- training.set.0[index, ]
val_data <- training.set.0[-index, ]

# Monitoreo del rendimiento en el conjunto de validación
   durante el entrenamiento
epochs <- 100
```

```
best_accuracy <- 0
best_epoch <- 0

# Función para calcular la precisión en el conjunto de
   validación (clasificación multiclase)
calculate_accuracy<- function(nn_model, validation_data) {
  predictions <- predict(nn_model, validation_data[, 1:6])
  #predictions = max.col(predictions)
  #predicted_classes <- apply(predictions, 1, which.max) - 1
     # Restamos 1 para obtener clases numeradas de 0 a 5
  predicted_classes = max.col(predictions)
  accuracy <- mean(predicted_classes == validation_data$etapa)
  return(accuracy)
}

for (epoch in 1:epochs) {
  nn_epoch <- neuralnet(formula = etapa ~ .,
                        data = train_data,
                        hidden = c(2),
                        threshold = 0.01,
                        linear.output = TRUE
  )

  accuracy <- calculate_accuracy(nn_epoch, val_data)
  cat("Epoch:", epoch, ", Validation Accuracy:", accuracy,
     "\n")

  # Guardar el modelo con mejor rendimiento en el conjunto de
     validación
  if (accuracy > best_accuracy) {
    best_accuracy <- accuracy
    best_epoch <- epoch
    best_nn <- nn_epoch
  } else {
```

```
    # Detener el entrenamiento si la precisión en el conjunto
       de validación comienza a disminuir
    if (epoch - best_epoch > 5) {  # Si la precisión no ha
       mejorado en las últimas 5 épocas
      cat("Early stopping at epoch:", epoch, "\n")
      break
    }
  }
}


# ---Entrenamiento

# Iniciar el temporizador
start_time <- Sys.time()

ann.net = neuralnet(formula = as.factor(etapa) ~ .,
                    data = training.set.0,
                    hidden = c(5),
                    threshold = 0.01,
                    linear.output = TRUE,
                    act.fct = "logistic",
                    rep=1)

# Detener el temporizador y mostrar el tiempo transcurrido
end_time <- Sys.time()
elapsed_time <- end_time - start_time
cat("Tiempo de entrenamiento:", as.numeric(elapsed_time),
   "segundos\n")

plot(ann.net, rep = "best")
#plotnet(ann.net)
head(ann.net$result.matrix)
```

```
data_pred = predict(ann.net, training.set.0, type = "response")
data_pred = max.col(data_pred)
table(data_pred)
table(training.set.0$etapa)

# Realizar el reemplazo según tus criterios
data_pred_t <- ifelse(data_pred == 1, 0,
                      ifelse(data_pred == 2, 1,
                             ifelse(data_pred == 3, 2,
                                    ifelse(data_pred == 4, 3,
                                           ifelse(data_pred ==
                                             5, 4,
                                            ifelse(data_pred
                                              == 6, 5,
                                              data_pred))))))

# Realizar el reemplazo según tus criterios
training.set.0_t <- ifelse(training.set.0$etapa == 1, 0,
                           ifelse(training.set.0$etapa == 2, 1,
                                  ifelse(training.set.0$etapa
                                    == 3, 2,
                                    ifelse(training.set.0$etapa
                                      == 4, 3,
                                       ifelse(training.set.0$etapa
                                         == 5, 4,
                                          ifelse(training.set.0$etapa==
                                            6, 5,
                                            training.set.0$etapa))))))

# Realizar el reemplazo según tus criterios
test.set.0_t <- ifelse(test.set.0$etapa == 1, 0,
                       ifelse(test.set.0$etapa == 2, 1,
                              ifelse(test.set.0$etapa == 3, 2,
                                     ifelse(test.set.0$etapa
```

```
                                        == 4, 3,
                                             ifelse(test.set.0$etapa
                                                == 5, 4,
                                             ifelse(test.set.0$etapa==
                                                6, 5,
                                                training.set.0$etapa))))))
cm.0 <- table(data_pred_t ,training.set.0_t)

matriz_confusion=cm.0

# Obtener nombres de clases
clases <- rownames(matriz_confusion)
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Inicializar vector para almacenar accuracies por clase
accuracies <- numeric(length(clases_evaluables))
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase

  # Imprimir el resultado para la clase actual
```

```
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
      "\n")
}


mc =
   confusionMatrix(as.factor(data_pred_t),as.factor(training.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad


# Supongamos que tienes las predicciones y las etiquetas
   reales en las variables y.pred.22 y test.set.2$etapa
set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(training.set.0_t),  # Las etiquetas reales
  pred = as.factor(data_pred_t)  # Las predicciones
)

# Calcular la matriz de confusión
cm <- conf_mat(truth_predicted, obs, pred)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)

# Graficar la matriz de confusión como un heatmap mejorado
autoplot(cm, type = "heatmap") +
  scale_fill_gradient(low = azul_crema_rgb, high =
      "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
```

```
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
          = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")

# ----- Prueba
data_pred.00 = predict(ann.net, test.set.0, type = "response")
data_pred.00 = max.col(data_pred.00)
table(data_pred.00)
table(test.set.0$etapa)

# Realizar el reemplazo según tus criterios
data_pred_t <- ifelse(data_pred.00  == 1, 0,
                      ifelse(data_pred.00  == 2, 1,
                             ifelse(data_pred.00  == 3, 2,
                                    ifelse(data_pred.00  == 4,
                                      3,
                                           ifelse(data_pred.00
                                             == 5, 4,
                                            ifelse(data_pred.00
                                             == 6, 5,
                                             data_pred.00
                                             ))))))
```

```
cm.00 <- table(data_pred_t,test.set.0_t)

matriz_confusion=cm.00
# Obtener nombres de clases
clases <- rownames(matriz_confusion)
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
# Inicializar vector para almacenar accuracies por clase
accuracies <- numeric(length(clases_evaluables))
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase

  # Imprimir el resultado para la clase actual
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}


mc =
```

```
   confusionMatrix(as.factor(data_pred_t),as.factor(test.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad


# Supongamos que tienes las predicciones y las etiquetas
   reales en las variables y.pred.22 y test.set.2$etapa
set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(test.set.0_t),
  pred = as.factor(data_pred_t )
)

cm.00 <- conf_mat(truth_predicted, obs, pred)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)

autoplot(cm.00, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
```

```
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Prueba",
       subtitle = "Valores Reales")


# ------------------------------------ Curva ROC Y AUC


# Instalar y cargar la librería pROC si aún no está instalada
if (!requireNamespace("pROC", quietly = TRUE)) {
  install.packages("pROC")
}
library(pROC)

# Ejemplo de etiquetas verdaderas y puntajes de predicción
   para un problema multiclase
num_classes <- 6

y_true  <- test.set.0$etapa reales
y_scores<- predict(ann.net, test.set.0, type = "response")


# Construir la curva ROC para cada clase y suavizarla
smoothed_roc_curves <- lapply(1:num_classes,
   function(class_index) {
  # Convertir a binario (uno-vs-rest)
  y_true_binary <- ifelse(y_true == class_index, 1, 0)
  # Calcular la curva ROC para esta clase
  roc_obj <- roc(y_true_binary, y_scores[, class_index])
```

```
  # Suavizar la curva ROC
  smooth(roc_obj, method = "density")
})


# Función para obtener el AUC de cada curva ROC
auc_class <- sapply(smoothed_roc_curves, function(roc_obj)
   auc(roc_obj))

# Imprimir el AUC para cada clase
print("AUC para cada clase:")
print(auc_class)

# Colores para las curvas ROC
colors <- rainbow(num_classes)

library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")

plot(smoothed_roc_curves[[1]], col = colors[1], main = "Curvas
   ROC Suavizada - Modelo 2",lwd=5)
for (i in 1:num_classes) {
  plot(smoothed_roc_curves[[i]], add = TRUE, col = colors[i],
     lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_class[i],
     3), "\n"), col = colors[i], pos = 1)  # Imprimir el AUC
     para las clases restantes
}

legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 1, lwd = 5, xjust = 1,yjust = 1)
```

Anexo 10: Tercer modelo ANN - 2 variables

```
# Metodo Holdout
training.set.2 = training.set.0
training.set.2 = training.set.2[,-1:-2]
training.set.2 = training.set.2[,-2]
training.set.2 = training.set.2[,-3]
training.set.0 = training.set.2

test.set.2 = test.set.0
test.set.2 = test.set.2[,-1:-2]
test.set.2 = test.set.2[,-2]
test.set.2 = test.set.2[,-3]
test.set.0 = test.set.2

# Iterar sobre los folds
for (i in 1:num_folds) {
# Dividir el conjunto de datos en entrenamiento y prueba según
   el fold actual
train_indices <- unlist(folds[-i])
test_indices <- folds[[i]]

training_set <- training.set.0[train_indices, ]
testing_set <- training.set.0[test_indices, ]

# 0.74
ann.net <- neuralnet(formula = as.factor(etapa) ~ .,
                     data = training_set,
                  hidden = c(5),
                   stepmax = 1e+05,
                threshold = 0.01,
                  linear.output = TRUE,
                act.fct = "tanh")


}
```

```r
# Dividir los datos en conjuntos de entrenamiento y validación
n <- nrow(training.set.0)
index <- sample(1:n, size = round(0.7 * n))  # 70% para
   entrenamiento, 30% para validación
train_data <- training.set.0[index, ]
val_data <- training.set.0[-index, ]

# Monitoreo del rendimiento en el conjunto de validación
   durante el entrenamiento
epochs <- 100
best_accuracy <- 0
best_epoch <- 0

# Función para calcular la precisión en el conjunto de
   validación (clasificación multiclase)
calculate_accuracy<- function(nn_model, validation_data) {
  validation_data_s=validation_data[, 1:2]
  predictions <- predict(nn_model, validation_data_s)

  predicted_classes = max.col(predictions)
  accuracy <- mean(predicted_classes == validation_data$etapa)
  return(accuracy)
}

for (epoch in 1:epochs) {
  nn_epoch <- neuralnet(formula = etapa ~ .,
                        data = train_data,
                        hidden = c(5),
                        threshold = 0.01,
                        linear.output = TRUE,
                        act.fct = "tanh"
  )
```

```
  accuracy <- calculate_accuracy(nn_epoch, val_data)
  cat("Epoch:", epoch, ", Validation Accuracy:", accuracy,
     "\n")

  # Guardar el modelo con mejor rendimiento en el conjunto de
     validación
  if (accuracy > best_accuracy) {
    best_accuracy <- accuracy
    best_epoch <- epoch
    best_nn <- nn_epoch
  } else {
    # Detener el entrenamiento si la precisión en el conjunto
       de validación comienza a disminuir
    if (epoch - best_epoch > 5) {  # Si la precisión no ha
       mejorado en las últimas 5 épocas
      cat("Early stopping at epoch:", epoch, "\n")
      break
    }
  }
}


# ------------ Entrenamiento

ann.net = neuralnet(formula = as.factor(etapa) ~ .,
                    data = training.set.0,
                    hidden = c(5),
                    threshold = 0.01,
                    linear.output = TRUE,
                    act.fct = "tanh"
                    ,rep=2)
plot(ann.net,type = "response")
```

```
data_pred = predict(ann.net, training.set.0, type = "response")
data_pred = max.col(data_pred)
table(data_pred)
table(training.set.0$etapa)


data_pred_t <- ifelse(data_pred == 1, 0,
                      ifelse(data_pred == 2, 1,
                             ifelse(data_pred == 3, 2,
                                    ifelse(data_pred == 4, 3,
                                           ifelse(data_pred ==
                                             5, 4,
                                                  ifelse(data_pred
                                                    == 6, 5,
                                                    data_pred))))))

training.set.0_t <- ifelse(training.set.0$etapa == 1, 0,
                           ifelse(training.set.0$etapa == 2, 1,
                                  ifelse(training.set.0$etapa
                                    == 3, 2,
                                      ifelse(training.set.0$etapa
                                        == 4,3,
                                      ifelse(training.set.0$etapa
                                        ==5, 4,
                                      ifelse(training.set.0$etapa==
                                        6, 5,
                                        training.set.0$etapa))))))

test.set.0_t <- ifelse(test.set.0$etapa == 1, 0,
                       ifelse(test.set.0$etapa == 2, 1,
                              ifelse(test.set.0$etapa == 3, 2,
```

```
                                        ifelse(test.set.0$etapa
                                           == 4, 3,
                                                ifelse(test.set.0$etapa
                                                   == 5, 4,
                                                ifelse(test.set.0$etapa==6,
                                                   5,training.set.0$etapa)))))
cm.0 <- table(data_pred_t ,training.set.0_t)

matriz_confusion=cm.0

clases <- rownames(matriz_confusion)
# Obtener las clases con al menos un TP o TN
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]

accuracies <- numeric(length(clases_evaluables))
# Calcular accuracy para cada clase evaluables
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  # Calcular el accuracy para la clase actual
  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  # Almacenar el accuracy en el vector
  accuracies[clase] <- accuracy_clase

  # Imprimir el resultado para la clase actual
  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
```

```
    "\n")
}

mc =
   confusionMatrix(as.factor(data_pred_t),as.factor(training.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad

set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(training.set.0_t),  # Las etiquetas reales
  pred = as.factor(data_pred_t)  # Las predicciones
)

cm <- conf_mat(truth_predicted, obs, pred)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)

autoplot(cm, type = "heatmap") +
  scale_fill_gradient(low = azul_crema_rgb, high =
     "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
```

```
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")

# --------------------- Prueba
data_pred.00 = predict(ann.net, test.set.0, type = "response")
data_pred.00 = max.col(data_pred.00)
table(data_pred.00)
table(test.set.0$etapa)


data_pred_t <- ifelse(data_pred.00  == 1, 0,
                      ifelse(data_pred.00  == 2, 1,
                             ifelse(data_pred.00  == 3, 2,
                                    ifelse(data_pred.00  == 4,
                                       3,
                                           ifelse(data_pred.00
                                              == 5, 4,
                                            ifelse(data_pred.00
                                              == 6,5,
                                              data_pred.00
                                              ))))))

cm.00 <- table(data_pred_t,test.set.0_t)

matriz_confusion=cm.00

clases <- rownames(matriz_confusion)
```

```
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]

accuracies <- numeric(length(clases_evaluables))

for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  accuracies[clase] <- accuracy_clase

  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
     "\n")
}

mc =
   confusionMatrix(as.factor(data_pred_t),as.factor(test.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad

set.seed(123)
truth_predicted <- data.frame(
```

```
  obs = as.factor(test.set.0_t),  # Las etiquetas reales
  pred = as.factor(data_pred_t )  # Las predicciones
)

cm.00 <- conf_mat(truth_predicted, obs, pred)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)

autoplot(cm.00, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz␣de␣Confusión␣-␣Prueba",
       subtitle = "Valores␣Reales")


# ------------------------------------ Curva ROC Y AUC


# Instalar y cargar la librería pROC si aún no está instalada
```

```r
if (!requireNamespace("pROC", quietly = TRUE)) {
  install.packages("pROC")
}
library(pROC)

num_classes <- 6

# Obtener las etiquetas reales y las probabilidades predichas
   para cada clase
y_true  <- test.set.0$etapa # Reemplaza test.set.0_t con tus
   etiquetas reales
y_scores<- predict(ann.net, test.set.0, type = "response")  #
   Reemplaza ann.net y test.set.0 con tus datos reales y
   predichos

smoothed_roc_curves <- lapply(1:num_classes,
   function(class_index) {
  # Convertir a binario (uno-vs-rest)
  y_true_binary <- ifelse(y_true == class_index, 1, 0)
  # Calcular la curva ROC para esta clase
  roc_obj <- roc(y_true_binary, y_scores[, class_index])
  # Suavizar la curva ROC
  smooth(roc_obj, method = "density")
})


auc_class <- sapply(smoothed_roc_curves, function(roc_obj)
   auc(roc_obj))

print("AUC␣para␣cada␣clase:")
print(auc_class)

colors <- rainbow(num_classes)
```

```
library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")

plot(smoothed_roc_curves[[1]], col = colors[1], main = "Curvas␣
   ROC␣Suavizada␣-␣Modelo␣2",lwd=5)
for (i in 1:num_classes) {
  plot(smoothed_roc_curves[[i]], add = TRUE, col = colors[i],
     lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_class[i],
     3), "\n"), col = colors[i], pos = 1)  # Imprimir el AUC
     para las clases restantes
}

legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 1, lwd = 5, xjust = 1,yjust = 1)
```

Anexo 11: Cuarto modelo ANN - 1 variable

```
# Metodo Holdout
training.set.0 = training.set.2
training.set.0 = training.set.0[,-2]

test.set.0 = test.set.2
test.set.0 = test.set.0[,-2]

# Iterar sobre los folds
for (i in 1:num_folds) {
# Dividir el conjunto de datos en entrenamiento y prueba según
   el fold actual
train_indices <- unlist(folds[-i])
test_indices <- folds[[i]]

training_set <- training.set.0[train_indices, ]
```

```
testing_set <- training.set.0[test_indices, ]

# Media: 0.72
ann.net <- neuralnet(formula = as.factor(etapa) ~ .,
                     data = training_set,
                    hidden = c(5),
                     stepmax = 1e+05,
                    threshold = 0.01,
                    linear.output = TRUE,
                   act.fct = "logistic")
}


# Dividir los datos en conjuntos de entrenamiento y validación
n <- nrow(training.set.0)
index <- sample(1:n, size = round(0.7 * n))  # 70% para
   entrenamiento, 30% para validación
train_data <- training.set.0[index, ]
val_data <- training.set.0[-index, ]

# Monitoreo del rendimiento en el conjunto de validación
   durante el entrenamiento
epochs <- 100
best_accuracy <- 0
best_epoch <- 0

calculate_accuracy<- function(nn_model, validation_data) {
  validation_data_s=validation_data[, 1]
  predictions <- predict(nn_model, validation_data_s)
  predicted_classes = max.col(predictions)
  accuracy <- mean(predicted_classes == validation_data$etapa)
  return(accuracy)
}
```

```
for (epoch in 1:epochs) {
  nn_epoch <- neuralnet(formula = etapa ~ .,
                        data = train_data,
                        hidden = c(5),
                        threshold = 0.01,
                        linear.output = TRUE,
                        act.fct = "logistic")


  accuracy <- calculate_accuracy(nn_epoch, val_data)
  cat("Epoch:", epoch, ", Validation Accuracy:", accuracy,
     "\n")

  if (accuracy > best_accuracy) {
    best_accuracy <- accuracy
    best_epoch <- epoch
    best_nn <- nn_epoch
  } else {
    # Detener el entrenamiento si la precisión en el conjunto
       de validación comienza a disminuir
    if (epoch - best_epoch > 5) {  # Si la precisión no ha
       mejorado en las últimas 5 épocas
      cat("Early stopping at epoch:", epoch, "\n")
      break
    }
  }
}

# ------------ Entrenamiento

ann.net = neuralnet(formula = as.factor(etapa) ~ .,
                    data = training.set.0,
                    hidden = c(5),
                    threshold = 0.01,
```

```
                    linear.output = TRUE,
                    act.fct = "logistic",
                    rep=1)

data_pred = predict(ann.net, training.set.0, type = "response")
data_pred = max.col(data_pred)
table(data_pred)
table(training.set.0$etapa)

data_pred_t <- ifelse(data_pred == 1, 0,
                      ifelse(data_pred == 2, 1,
                             ifelse(data_pred == 3, 2,
                                    ifelse(data_pred == 4, 3,
                                           ifelse(data_pred ==
                                             5, 4,
                                                  ifelse(data_pred
                                                    == 6, 5,
                                                    data_pred))))))

training.set.0_t <- ifelse(training.set.0$etapa == 1, 0,
                           ifelse(training.set.0$etapa == 2, 1,
                                  ifelse(training.set.0$etapa
                                    == 3, 2,
                                         ifelse(training.set.0$etapa
                                           == 4,3,
                                    ifelse(training.set.0$etapa
                                      == 5, 4,
                                    ifelse(training.set.0$etapa==
                                      6, 5,
                                      training.set.0$etapa))))))

test.set.0_t <- ifelse(test.set.0$etapa == 1, 0,
                       ifelse(test.set.0$etapa == 2, 1,
                              ifelse(test.set.0$etapa == 3, 2,
```

```
                                        ifelse(test.set.0$etapa
                                          == 4, 3,
                                               ifelse(test.set.0$etapa
                                                  == 5, 4,
                                               ifelse(test.set.0$etapa==
                                                  6,
                                                  5,training.set.0$etapa)))))
cm.0 <- table(data_pred_t ,training.set.0_t)

nueva_fila <- matrix(0, nrow = 2, ncol = ncol(cm.0))

cm.0_actualizada <- rbind(cm.0[1,], nueva_fila, cm.0[-1,])

rownames(cm.0_actualizada) <- c("0", "1", "2", "3", "4", "5")

print(cm.0_actualizada)

matriz_confusion=cm.0_actualizada

clases <- rownames(matriz_confusion)
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
accuracies <- numeric(length(clases_evaluables))
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)
```

```
  accuracies[clase] <- accuracy_clase

  cat("Accuracy para la Clase", clase, ":", accuracy_clase,
      "\n")
}

data_pred_t <- factor(data_pred_t, levels =
   c(levels(data_pred_t), "0", "1","2","3","4","5"))

mc =
   confusionMatrix(as.factor(data_pred_t),as.factor(training.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad

set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(training.set.0_t),  # Las etiquetas reales
  pred = as.factor(data_pred_t)  # Las predicciones
)

cm <- conf_mat(truth_predicted, obs, pred)
azul_crema_rgb <- rgb(207, 221, 233, maxColorValue = 255)

autoplot(cm, type = "heatmap") +
  scale_fill_gradient(low = azul_crema_rgb, high =
     "steelblue", na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
```

```
        axis.text.x = element_text(angle = 45, hjust = 1, size
          = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Entrenamiento",
       subtitle = "Valores Reales")

# ----------------- Prueba
data_pred.00 = predict(ann.net, test.set.0, type = "response")
data_pred.00 = max.col(data_pred.00)
table(data_pred.00)
table(test.set.0$etapa)

# Realizar el reemplazo según tus criterios
data_pred_t <- ifelse(data_pred.00  == 1, 0,
                      ifelse(data_pred.00  == 2, 1,
                             ifelse(data_pred.00  == 3, 2,
                                    ifelse(data_pred.00  == 4,
                                      3,
                                           ifelse(data_pred.00
                                              == 5, 4,
                                                  ifelse(data_pred.00
                                                    == 6, 5,
                                                    data_pred.00
                                                    ))))))
```

```
cm.00 <- table(data_pred_t,test.set.0_t)

nueva_fila <- matrix(0, nrow = 2, ncol = ncol(cm.00))

cm.00_actualizada <- rbind(cm.00[1,], nueva_fila, cm.00[-1,])

rownames(cm.00_actualizada) <- c("0", "1", "2", "3", "4", "5")

print(cm.00_actualizada)

matriz_confusion=cm.00_actualizada
clases <- rownames(matriz_confusion)
clases_evaluables <- clases[apply(matriz_confusion, 1,
   function(x) sum(x) > 0)]
accuracies <- numeric(length(clases_evaluables))
for (clase in clases_evaluables) {
  TP <- matriz_confusion[clase, clase]
  TN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, colnames(matriz_confusion) != clase])
  FP <- sum(matriz_confusion[clase, colnames(matriz_confusion)
     != clase])
  FN <- sum(matriz_confusion[rownames(matriz_confusion) !=
     clase, clase])

  accuracy_clase <- (TP + TN) / (TP + TN + FP + FN)

  accuracies[clase] <- accuracy_clase

  cat("Accuracy␣para␣la␣Clase", clase, ":", accuracy_clase,
     "\n")
}

data_pred_t <- factor(data_pred_t, levels =
```

```
   c(levels(data_pred_t), "0", "1","2","3","4","5"))

mc =
   confusionMatrix(as.factor(data_pred_t),as.factor(test.set.0_t))
mc1 = t(mc$table) # Matriz de Confusion

acc_total = mc$overall[1] # Accuracy General
prec = mc$byClass[,5] # Precision
F.measure = mc$byClass[,7] # F-measure
sensi = mc$byClass[,1] # Sensibilidad
especi = mc$byClass[,2] # Especificidad

set.seed(123)
truth_predicted <- data.frame(
  obs = as.factor(test.set.0_t),
  pred = as.factor(data_pred_t)
)

cm.00 <- conf_mat(truth_predicted, obs, pred)
verde_crema_rgb <- rgb(245, 255, 250, maxColorValue = 255)
verde_rgb <- rgb(0, 128, 0, maxColorValue = 255)

autoplot(cm.00, type = "heatmap") +
  scale_fill_gradient(low = verde_crema_rgb, high = verde_rgb,
     na.value = "grey90") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 45, hjust = 1, size
           = 14),
        axis.text.y = element_text(size = 14),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
```

```
        panel.background = element_blank(),
        legend.position = "right",
        plot.title = element_text(hjust = 0.5, size = 20),
        plot.subtitle = element_text(hjust = 0.5,size = 14),
        plot.caption = element_text(size = 14)) +
  guides(fill = guide_colorbar(title = "Frecuencia")) +
  labs(title = "Matriz de Confusión - Prueba",
       subtitle = "Valores Reales")

# ------------------------- Curva ROC Y AUC

if (!requireNamespace("pROC", quietly = TRUE)) {
  install.packages("pROC")
}
library(pROC)
 multiclase
num_classes <- 6

y_true  <- test.set.0$etapa
y_scores<- predict(ann.net, test.set.0, type = "response")

smoothed_roc_curves <- lapply(1:num_classes,
   function(class_index) {
  y_true_binary <- ifelse(y_true == class_index, 1, 0)
  roc_obj <- roc(y_true_binary, y_scores[, class_index])
  smooth(roc_obj, method = "density")
})


auc_class <- sapply(smoothed_roc_curves, function(roc_obj)
   auc(roc_obj))

print("AUC para cada clase:")
print(auc_class)
```

```r
colors <- rainbow(num_classes)

library(paletteer)
colors <- paletteer_d("ggsci::nrc_npg")
plot(smoothed_roc_curves[[1]], col = colors[1], main = "Curvas␣
   ROC␣Suavizada␣-␣Modelo␣2",lwd=5)
for (i in 1:num_classes) {
  plot(smoothed_roc_curves[[i]], add = TRUE, col = colors[i],
     lwd = 5)
  text(0.2, 0.5 - i * 0.05, paste("AUC:", round(auc_class[i],
     3), "\n"), col = colors[i], pos = 1)
}

legend("bottomright", legend = 1:num_classes-1, col = colors,
   lty = 1, cex = 1, lwd = 5, xjust = 1,yjust = 1)
```