Universidad Internacional de La Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia Artificial

Clasificador con redes neuronales para el pronóstico de la enfermedad renal crónica en la población colombiana

Trabajo Fin de Máster

Presentado por: Vásquez Morales, Gabriel Ricardo

Director: Martínez Monterrubio, Sergio Mauricio. PhD

Ciudad: Bogotá, Colombia
Fecha: 24 de julio de 2019

Índice de contenido

Índice de figuras

Índice de tablas

Resumen

Este trabajo tiene como objetivo la creación de un clasificador basado en redes neuronales para pronosticar si una persona está en riesgo de desarrollar enfermedad renal crónica (ERC). El modelo se entrenó con los datos demográficos y la información de las atenciones médicas de dos grupos poblacionales: por una parte, personas diagnosticadas con ERC en Colombia durante el año 2018, y por otra, una muestra de personas sin diagnóstico de esta enfermedad. Una vez entrenado el modelo y aplicadas las métricas de evaluación para algoritmos de clasificación, el modelo alcanzó 95% de exactitud en el conjunto de datos de prueba, siendo por lo tanto viable su aplicación para el pronóstico de la enfermedad. Tras su implementación, se identificaron 3.494.516 personas en riesgo de desarrollar ERC en Colombia, es decir un 7% del total de la población.

Palabras Clave: clasificador, enfermedad renal crónica, pronóstico, red neuronal.

Abstract

This paper aims to create a neural network-based classifier to predict whether a person is at risk of developing chronic kidney disease (CKD). The model was trained with the demographic data and medical care information of two population groups: on the one hand, people diagnosed with CKD in Colombia during 2018, and on the other, a sample of people without a diagnosis of this disease. Once the model was trained and evaluation metrics for classification algorithms were applied, the model achieved 95% accuracy in the test data set, making its application for disease prognosis feasible. After implementation, 3,494,516 people were identified as being at risk of developing CKD in Colombia, or 7% of the total population.

Keywords: chronic kidney disease, classifier, forecast, neural network.

Capítulo 1 – Introducción

La enfermedad renal crónica (ERC) es una de las patologías que más costos genera al sistema de salud colombiano. El costo acumulado del tratamiento de los pacientes con ERC con corte al año 2017 fue de 3.241.558.048.735 pesos colombianos (aproximadamente 856.170.837 euros) (Ministerio de Salud y Protección Social, 2019). Esta enfermedad produce disminución de la calidad de vida de los pacientes y puede llegar a provocar la muerte. Durante al año 2017 fallecieron 33.472 personas con ERC, es decir una mortalidad de 67,9 personas por cada 100.000 habitantes (Fondo Colombiano de Enfermedades de Alto Costo, 2017). La *Figura 1* muestra el incremento en el número de muertes en pacientes con ERC en los últimos años.

Muchas personas no acuden a los servicios de salud a tiempo y cuando lo hacen la enfermedad ya se encuentra en una etapa avanzada. Por esta razón se desarrolló un modelo predictivo basado en redes neuronales para identificar si una persona está en riesgo de presentar ERC. Este modelo se entrenó con los datos de los Registros Individuales de Prestación de Servicios en Salud (RIPS). Una vez entrenada la red neuronal y aplicadas las pruebas de validación del modelo, fue posible crear un mapa del riesgo en la población. Esto con el objetivo de generar un sistema de alertas individual que permita realizar las actividades de prevención necesarias. De esta manera es posible evitar que desarrollen la enfermedad, o confirmar el diagnóstico y brindarles el tratamiento necesario.

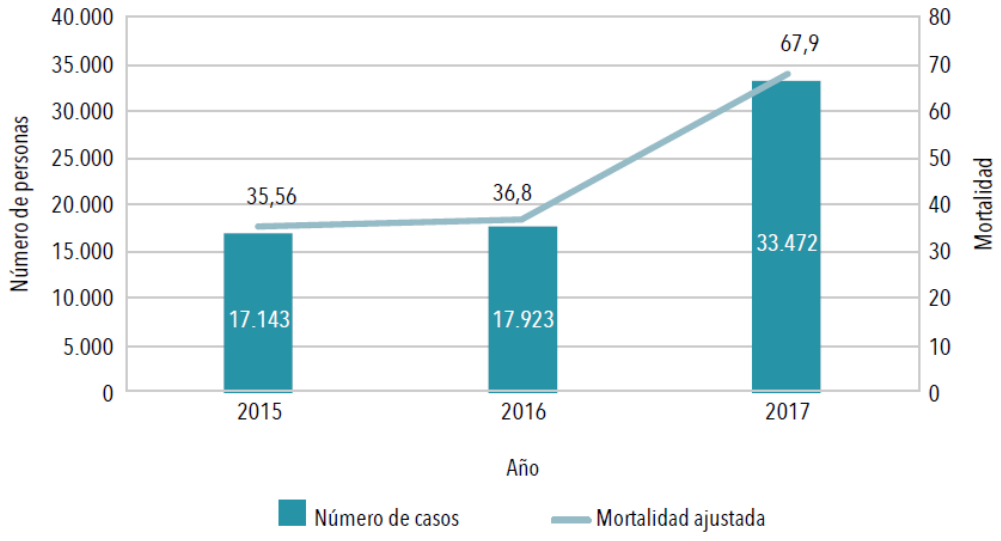


Figura 1: Mortalidad por ERC. *(*Fondo Colombiano de Enfermedades de Alto Costo, 2017)

1.1 Motivación

La enfermedad renal crónica consiste en una disminución progresiva e irreversible de la función renal, que trae como consecuencia la incapacidad de los riñones para eliminar los desechos a través de la orina (Fondo Colombiano de Enfermedades de Alto Costo, 2014). Esta patología afecta aproximadamente a un 10% de la población mundial, siendo una enfermedad silenciosa que no presenta síntomas hasta que ya se encuentra en un estado avanzado (Organización Panamericana de la Salud, 2015). El tratamiento para las personas que han desarrollado la enfermedad consiste en realizar una diálisis o filtrado de la sangre con el fin de eliminar los desechos peligrosos que el riñón ya no es capaz de desechar. Este tratamiento se puede realizar de dos formas: a través de hemodiálisis, en la cual el paciente debe acudir varias veces por semana al centro médico para ser tratado, o a través de diálisis peritoneal, donde se realiza una cirugía para introducir un catéter en el abdomen del paciente y de esta manera el mismo se puede administrar el tratamiento en su domicilio. La alternativa a la diálisis consiste en realizar un trasplante de riñón, aunque esta es una solución que puede implicar mucho tiempo de espera para el paciente, debido a las dificultades para la donación de órganos. El tratamiento para la enfermedad renal crónica además de ser altamente invasivo e impactar negativamente en la calidad de vida del paciente, también es bastante costoso para los sistemas de salud (Fondo Colombiano de Enfermedades de Alto Costo, 2017).

Diferentes entidades internacionales, como la Organización Mundial de la Salud, han hecho llamados a los gobiernos locales y a la comunidad científica internacional para prevenir la enfermedad renal crónica y mejorar el acceso a su tratamiento, de manera que se reduzca la brecha que separa a los pacientes del tratamiento que puede prolongar y salvar su vida. Entre las estrategias que señalan se encuentran mejorar la detección temprana de la enfermedad y fortalecer el conocimiento de la situación en cada país. (Organización Panamericana de la Salud, 2015).

En Colombia, de acuerdo con datos oficiales reportados por la Cuenta de Alto Costo, para el año 2017 había 1.406.364 personas con Enfermedad Renal Crónica, esto representa una prevalencia de 2,9 casos por cada 100 habitantes (Fondo Colombiano de Enfermedades de Alto Costo, 2017). Esta cifra ha ido en aumento en los últimos años, tal como se puede ver en la *Figura 2*.

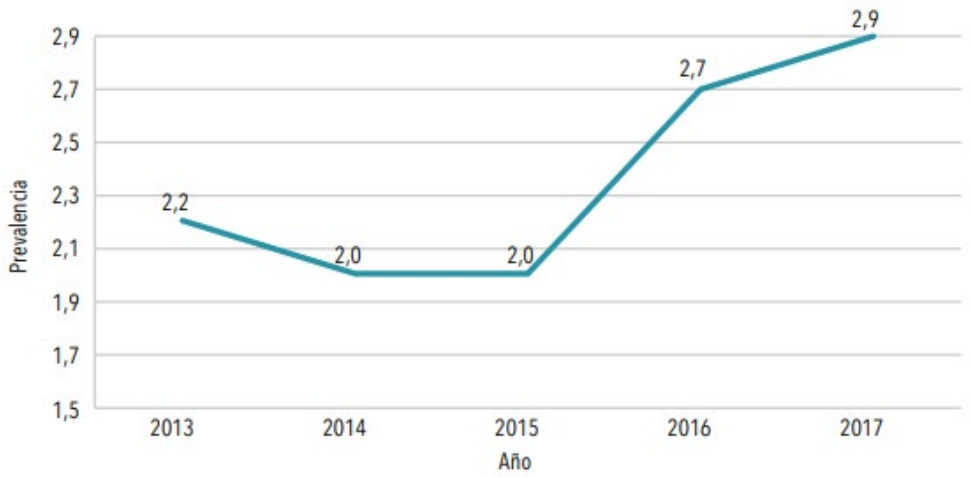


Figura 2: Prevalencia de ERC en Colombia. (Fondo Colombiano de Enfermedades de Alto Costo, 2017)

1.2 Planteamiento del problema

En la actualidad las redes neuronales, y en general los algoritmos de aprendizaje automático han comenzado a utilizarse como herramientas de apoyo para el diagnóstico y detección de enfermedades. Estos algoritmos se entrenan con grandes volúmenes de datos, que incluyen imágenes diagnósticas, pruebas de laboratorio e historias clínicas. A partir de los datos de entrada el algoritmo encuentra patrones que permiten identificar aquellos casos en los que se presenta alguna patología específica y los generaliza para la detección de nuevos pacientes. Las redes neuronales han demostrado tener un mejor desempeño en tareas de clasificación que otros modelos de aprendizaje automático tradicionales. (West y West, 2000).

En el presente trabajo se propone el uso de una red neuronal para el pronóstico de la enfermedad renal crónica en la población colombiana. Dicho modelo se entrenó con la información disponible en las bases de datos del ministerio de salud y protección social sobre prestación de servicios de salud en Colombia durante los años 2009 a 2018. Específicamente se tomó el conjunto de datos demográficos y de atenciones en salud correspondientes a personas diagnosticadas con ERC en el año 2018, así como una muestra de personas que no presentaban la patología. El modelo debía ser capaz de identificar los patrones relacionados con la presencia de la ERC y de esta forma aplicarlos en la identificación de nuevos casos.

1.3 Estructura del trabajo

En el siguiente capítulo se describe el estado del arte de las técnicas de aprendizaje automático aplicadas al sector salud, especialmente en las que se utilizan redes neuronales para el diagnóstico de enfermedades. En el capítulo 3 se describen los procesos realizados para diseñar, modelar e implementar la red neuronal siguiendo los pasos de la metodología CRISP-DM. En la comprensión del negocio se exponen los principales objetivos y necesidades del estado colombiano en relación con el diagnóstico de enfermedades haciendo uso de tecnologías informáticas. También se identifican las bases de datos disponibles y se priorizan las variables que se emplearán en el proyecto.

La captura y exploración de los datos abarca las técnicas empleadas para obtener los datos necesarios para el entrenamiento y pruebas del modelo de redes neuronales a partir de las bases de datos del ministerio de salud en Colombia, así como una exploración inicial de los mismos para obtener conocimiento preliminar e identificar posibles problemas de calidad. El subcapítulo de limpieza y preparación de datos describe el proceso de transformación de los datos, incluyendo selección de variables o características, unión de tablas, transformación de filas a columnas, manejo de variables categóricas, tratamiento de valores vacíos y demás operaciones de limpieza requeridas para obtener el conjunto de datos final.

En el modelado de la red neuronal se hace una introducción al concepto de redes neuronales y aprendizaje profundo, así como al proceso de entrenamiento haciendo uso del algoritmo de *backpropagation*, el uso de función de coste y gradiente descendente. Se describe el proceso de separación de conjuntos de datos para entrenamiento, validación y pruebas, y se indican los pasos seguidos para realizar el entrenamiento de la red neuronal. Se indican las técnicas empleadas para la optimización de hiperparámetros y la obtención del modelo final. La evaluación del modelo describe el proceso de pruebas del modelo y las métricas empleadas para la evaluación del mismo. Se discuten los resultados obtenidos y la viabilidad de su implementación.

Lugo se realiza entrenamiento del conjunto de datos con otros modelos de aprendizaje automático como lo son: SVM y Random Forest, con el fin de comparar la precisión obtenida frente al modelo de redes neuronales. A continuación, se realiza la implementación del modelo, durante la cual se describe el uso del modelo entrenado con redes neuronales para realizar el pronóstico de la enfermedad renal crónica usando el conjunto completo de datos de la población colombiana. Se indica el número de personas nuevas para las que el algoritmo identifica riesgo de padecer la enfermedad y se analiza su distribución demográfica.

Finalmente se presentan las conclusiones y trabajo futuro. Se discuten los hallazgos encontrados a la luz de los objetivos propuestos para el proyecto, así como la viabilidad de su uso para el pronóstico de otras enfermedades.

1.4 Hipótesis

1.4.1 Hipótesis Nula (H0)

De acuerdo con las guías de práctica clínica sobre Enfermedad Renal Crónica, el diagnóstico de esta patología requiere la toma y análisis de muestras de sangre y/o orina del paciente, con el fin de identificar el valor de la tasa de filtración glomerular (TFG), que determina el grado de funcionamiento de los riñones de la persona (Ministerio de Salud y Protección Social, 2016). Valores de TFG menores a <60 indican la presencia de enfermedad renal crónica debido a la pérdida de al menos la mitad de la función renal (Flores et al., 2009). En el presente trabajo se establece como hipótesis nula el hecho que:

H_0: Solo se pueda diagnosticar la ERC a partir de las pruebas de laboratorio que permitan determinar la tasa de filtración glomerular (TFG).

1.4.2 Hipótesis Alternativa (HA)

Las técnicas de aprendizaje automático han demostrado tener una gran capacidad para realizar el diagnóstico de diferentes enfermedades, entre estas la enfermedad renal crónica (Xiao et al., 2019), sin embargo, dentro de los parámetros empleados para entrenar los modelos se utilizan principalmente los resultados de pruebas de laboratorio. Los registros de atenciones en salud que tiene el ministerio de salud de Colombia cuentan con el historial de patologías diagnosticadas a los pacientes a nivel nacional, aunque no dispone de los valores correspondientes a los resultados de pruebas de laboratorio. Es por esto que el presente trabajo busca entrenar un modelo predictivo capaz de pronosticar la presencia de ERC en una persona sin tener en cuenta información de pruebas diagnósticas, sino únicamente con los datos demográficos de la persona y su historial de diagnósticos de enfermedades previas. De esta manera la hipótesis que se busca demostrar es la siguiente:

H_A: Un modelo basado en redes neuronales es capaz de pronosticar el riesgo de desarrollar enfermedad renal crónica a partir de los registros individuales de prestación de servicios de salud de un paciente con igual o mayor precisión que los resultados de pruebas de laboratorio.

1.4.3 Preguntas de investigación

Las preguntas de investigación a la que este trabajo busca dar respuesta son las siguientes:

1. ¿Puede una red neuronal pronosticar si un ciudadano colombiano está en riesgo de desarrollar enfermedad renal crónica a partir de sus registros individuales de prestación de servicios (RIPS)?
2. ¿Es mejor un modelo de red neuronal que los algoritmos de SVM y Random Forest para realizar el pronóstico de la enfermedad renal crónica?
3. ¿Cuántos ciudadanos colombianos a quienes se les ha prestado servicios de salud y aún no se les ha diagnosticado ERC se encuentran en riesgo de desarrollarla?

Para el caso de la última pregunta se debe tener en cuenta que la población total de Colombia, de acuerdo con las proyecciones del Departamento Administrativo Nacional de Estadística (DANE), es de 49.834.240 para el año 2018 (DANE, 2019), pero el número de personas para las que se cuenta con información de atenciones en salud en la base de datos de RIPS en el periodo 2015-2018 es de 39.277.086.

1.5 Objetivos

1.5.1 Objetivo general

Construir una red neuronal que permita pronosticar el riesgo de desarrollar enfermedad renal crónica a partir de la información reportada en los Registros Individuales de Prestación de Servicios (RIPS), obteniendo una métrica de precisión superior al 90%.

1.5.2 Objetivos específicos

Los objetivos específicos de este trabajo son los siguientes:

1. Identificar el algoritmo de aprendizaje automático con mayor capacidad para el pronóstico de la enfermedad renal crónica a partir del conjunto de datos demográficos y de atenciones médicas obtenidos de la base de datos RIPS.
2. Implementar el modelo de redes neuronales para el pronóstico de la ERC en la población colombiana haciendo uso de los registros médicos disponibles en la base de datos de RIPS.
3. Analizar el perfil demográfico de los pacientes a quienes la red neuronal pronostica con alto riesgo de desarrollar enfermedad renal crónica.

Capítulo 2 – Estado del arte

Este capítulo se divide en cuatro secciones. En la introducción se revisará el concepto de aprendizaje automático, mencionando los principales algoritmos que se emplean para entrenamiento de clasificadores. La segunda parte se enfocará específicamente en los clasificadores con redes neuronales, el concepto de aprendizaje profundo y las principales herramientas existentes en la actualidad para la creación de estos modelos. Una tercera parte tratará el uso de técnicas de aprendizaje automático en el diagnóstico de enfermedades y la cuarta y última parte hará un recorrido por diferentes investigaciones en las que se han desarrollado redes neuronales para el pronóstico de la enfermedad renal crónica.

2.1 Aprendizaje automático

El aprendizaje automático o machine learning es un campo de aplicación de la inteligencia artificial que tiene como propósito la creación de algoritmos que permitan a las máquinas aprender de un conjunto de datos y realizar tareas de forma inteligente sin que hayan sido previamente programadas con un conjunto de reglas específicas. Un ejemplo típico de aprendizaje automático son los sistemas antispam, que se entrenan con un gran volumen de datos obtenidos de correos electrónicos previamente etiquetados como spam o no spam. El algoritmo debe encontrar patrones en el texto de los mensajes que le permitan identificar si un mensaje de correo se puede clasificar como spam o no (Tretyakov, 2004). El aprendizaje automático es un campo cuyos orígenes se pueden remontar hasta la década de 1950, con investigaciones como las de Arthur Samuel, quien diseño el primer programa para computadora capaz de aprender (Samuel, 1959). Este programa jugaba a las damas y era capaz de aprender de las partidas anteriores. Después de varios años de grandes logros y entusiasmo por parte de los investigadores, el aprendizaje automático fue perdiendo popularidad, hasta que en la primera década del siglo XXI adquirió de nuevo un papel relevante en los estudios sobre la inteligencia artificial. Esto se debe principalmente a dos factores que han impulsado su rápido crecimiento hasta ahora: por una parte, la aparición de internet y el aumento en la cantidad de información disponible para alimentar los modelos de aprendizaje automático, y por otra parte las mejoras en los recursos de hardware y por lo

tanto en la capacidad de computo necesaria para procesar de forma eficiente estos grandes volúmenes de datos.

2.1.1 Tipos de aprendizaje automático

Los algoritmos de aprendizaje automático se pueden dividir en dos grandes grupos: aprendizaje supervisado y aprendizaje no supervisado. El aprendizaje supervisado utiliza datos etiquetados, es decir que se le indica al modelo cual es la salida esperada para cada ejemplo con el objetivo de que encuentre los patrones que relacionan las variables de entrada con la de salida. En aprendizaje no supervisado no se dispone de ejemplos etiquetados y el modelo debe aprender a agruparlos de acuerdo con los valores de las variables de entrada. A su vez el aprendizaje supervisado se puede subdividir en problemas de regresión y problemas de clasificación, mientras que el aprendizaje no supervisado se puede dividir en problemas de agrupamiento o clusterización, y problemas de detección de anomalías (James, Witten, Hastie, y Tibshirani, 2013).

2.1.2 Problemas de clasificación

El objetivo de los algoritmos de clasificación es catalogar cada ejemplo en dos o varias categorías, dependiendo del número de clases definidas. Si solo hay dos categorías, como en el caso del detector de spam donde cada ejemplo solo puede ser spam o no spam se habla de un clasificador binario, mientras que, si existen varias etiquetas, como puede ser el caso de un algoritmo que identifique dígitos escritos a mano, se trata de un clasificador multiclase. El algoritmo se entrena con un conjunto de ejemplos en los que se le indica cual es la clase o etiqueta esperada, con el propósito que el modelo encuentre patrones entre las variables de entrada y la clase de salida, sea capaz de aprender estos patrones y generalizarlos para clasificar nuevos ejemplos para los que se quiere hallar la etiqueta correspondiente.

2.1.3 Conjuntos de entrenamiento y pruebas

Uno de los aspectos más importantes en los modelos de aprendizaje automático es su capacidad para clasificar correctamente nuevos ejemplos. Si un modelo es capaz de clasificar correctamente los ejemplos con los que es entrenado, pero luego no es capaz de clasificar correctamente nuevos ejemplos, no es útil ya que no logra generalizar el conocimiento aprendido. Es por esto que la creación de un modelo de aprendizaje automático se divide en dos fases, como se muestra en la Figura 3. La primera fase es el entrenamiento del modelo, en el que el algoritmo de aprendizaje automático toma los datos

de entrada junto con la etiqueta o clase del ejemplo y genera un modelo de clasificación. La segunda fase, denominada de predicción o pruebas, utiliza como entrada un nuevo conjunto de datos que se procesa utilizando el modelo obtenido en la fase de entrenamiento para obtener la etiqueta más probable para cada instancia. Para evaluar el desempeño del modelo se comparan las etiquetas generadas contra las etiquetas reales de cada ejemplo y se obtiene una métrica que determina si el modelo es viable o no. El proceso de dividir los datos entre los que se van a utilizar para entrenamiento y los que se utilizar para el test se conoce como hold-out, y normalmente se utiliza un 80 % de los datos para crear el conjunto de entrenamiento y un 20 % de los datos para generar el conjunto de datos de test.

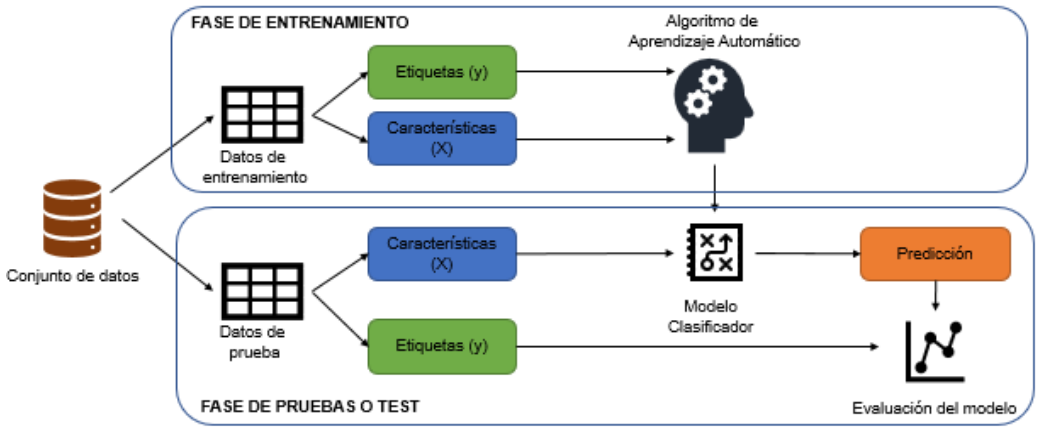


Figura 3: Ejemplo de entrenamiento y test en aprendizaje automático.
Fuente: Elaboración propia.

Una porción de los datos de entrenamiento se puede utilizar para validación cruzada. La validación cruzada permite evaluar el desempeño del modelo mientras se entrena. Esta técnica divide de forma aleatoria el conjunto de datos en un número k de particiones y por cada partición realiza una iteración. Durante cada iteración se toman los datos de la partición como conjunto de test y el resto de ejemplos como conjunto de entrenamiento (James et al., 2013). De esta manera un ejemplo siempre se va a utilizar tanto para entrenamiento como para validación.

2.1.4 Principales algoritmos de clasificación

Existen diferentes algoritmos de aprendizaje supervisado que se han venido usando en tareas de clasificación tales como Regresión Logística, Naive Bayes, Máquinas de Vectores de Soporte (SVM), Arboles de decisión, Random Forest, Redes Neuronales, etc.

Las máquinas de vectores de soporte o support vector machines (SVM) es uno de los primeros algoritmos de aprendizaje automático que no se basan en modelos estadísticos,

sino en modelos geométricos que se resuelven mediante un problema de optimización con restricciones. El objetivo de las máquinas de vectores de soporte es buscar un hiperplano que separe los datos de acuerdo con las clases del problema.

Random Forest es una técnica que emplea, por un lado, combinación o bagging de árboles de decisión, junto con selección de variables aleatorias. Esto permite que el modelo pueda trabajar con un gran número de variables de entrada a la vez que resuelve el problema de baja capacidad predictiva de los árboles de decisión. El modelo de Random Forest es bastante popular y permite trabajar con grandes volúmenes de datos obteniendo resultados con un alto nivel de precisión.

En la siguiente sección se describirá el funcionamiento de las redes neuronales y las principales herramientas para su construcción.

2.2 Redes neuronales y aprendizaje profundo

Las redes neuronales artificiales modelan una función entre un conjunto de variables de entrada y una clase o variable de salida, emulando el funcionamiento de las neuronas en el cerebro (McCulloch y Pitts, 1943). Para lograr este objetivo, las redes de neuronas emplean nodos interconectados entre sí. Una neurona biológica obtiene impulsos de otras neuronas a través de las dendritas y transmite a su vez un impulso a otras neuronas a través del axón. Esto es equivalente a las variables de entrada y salida de los modelos de aprendizaje automático. De manera similar al cerebro, las redes neuronales se componen de muchas neuronas y sus conexiones entre sí.

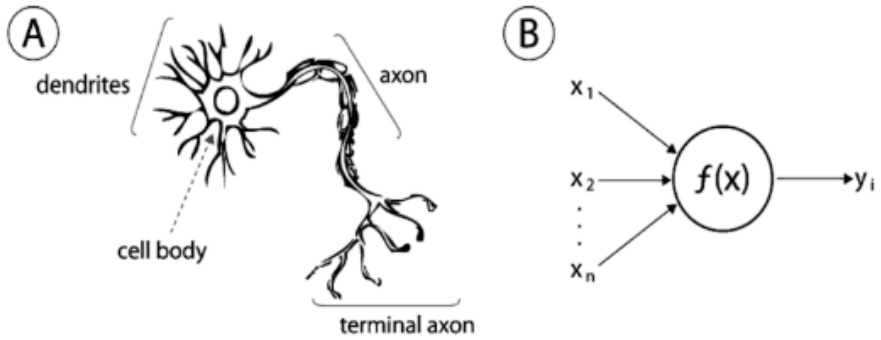


Figura 4: Similitud entre una neurona biológica y una artificial. (A) Neurona humana; (B) Neurona artificial. (Maltarollo, Honório y da Silva, 2013)

En una neurona artificial los valores de la señal de entrada se multiplican por los pesos (W) o parámetros de la red. Luego se suman los productos y se utiliza una función matemática

de activación para generar la salida de la neurona, que es enviada a otras neuronas de la red. Las redes neuronales se han popularizado debido al incremento en la capacidad de los recursos computacionales necesarios para procesar grandes volúmenes de información. Son modelos muy versátiles que se pueden utilizar en diferentes tareas de aprendizaje automático, como regresión y clasificación. Su uso ha permitido avanzar en campos como visión artificial y procesamiento de lenguaje natural. Una red neuronal que tenga por lo menos una capa oculta es capaz de aproximar cualquier función matemática, es decir que actúa como un aproximador universal (Hornik, Stinchcombe y White, 1989).

Un modelo de redes neuronales debe tener una capa de entrada (variables de entrada), una capa de salida (clase o clases a predecir) y una o más capas ocultas (donde la red relaciona las variables de entrada con las de salida), como se muestra en la Figura 4. Si la red cuenta con varias capas ocultas o intermedias esto hace que el modelo tenga una mayor capacidad de representación e interpretación de los datos. Esto se conoce como aprendizaje profundo, o *deep learning* (Hinton, Osindero y Teh, 2006), ya que permite crear modelos que son capaces de interpretar datos de naturaleza compleja como imágenes y sonidos, con un mayor grado de precisión que otros modelos de aprendizaje automático.

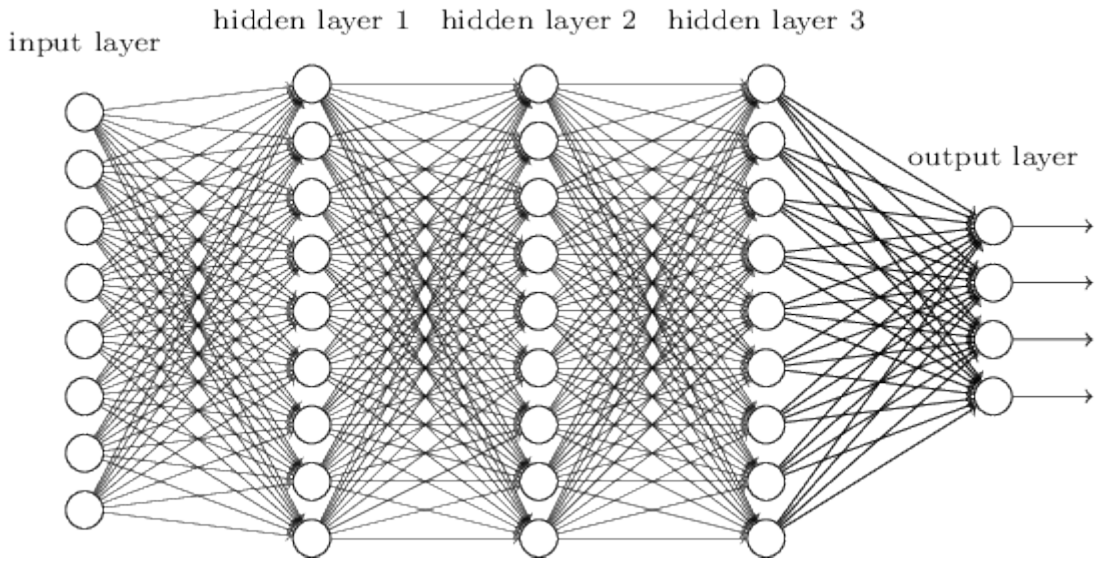


Figura 5: Capas de una red neuronal. (Nielsen, 2015)

2.2.1 Algoritmo de backpropagation

El proceso de entrenamiento de una red neuronal consiste en encontrar el valor óptimo de los pesos o parámetros de la red. Se trata de una tarea costosa computacionalmente debido a que implica el cálculo de gradientes o derivadas. El algoritmo empleado para el entrenamiento de la red neuronal se conoce con el nombre de backpropagation (Rumelhart,

Hinton y Williams, 1986). El algoritmo funciona de forma iterativa y se desarrolla en dos fases durante cada ciclo:

Hacia adelante (forward): se multiplica el valor de las entradas de la red por los pesos calculados, se aplica la función de activación y se avanza hasta alcanzar la capa de salida de la red.

Hacia atrás (backward): se comparan los valores de salida de la red con los valores esperados y se obtiene una medida del error. Este valor del error o *loss* se propaga hacia atrás en la red. A partir de la medida del error se ajustan los pesos de la red y se inicia otro ciclo del algoritmo.

Para optimizar los pesos de una red se utiliza una técnica denominada descenso del gradiente (gradient descent). Los pesos de la red se modifican siguiendo la dirección que produce una mayor reducción del error, utilizando para ello el cálculo del gradiente en cada nodo. Este valor se obtiene multiplicando el gradiente propagado desde la capa de salida de la red, por el gradiente local respecto a cada entrada del nodo.

2.2.2 Herramientas para la creación de redes neuronales

Python: es un lenguaje de programación de uso general, que figura entre los más populares a la hora de desarrollar prototipos basados en inteligencia artificial en general y aprendizaje automático en particular. Python es un lenguaje interpretado con una sintaxis sencilla que permite desarrollar y evaluar de forma iterativa y ágil modelos analíticos. Es, además, un producto de código abierto y de uso gratuito.

R: es un lenguaje de programación enfocado en el análisis estadístico. Es de uso libre y posee un gran conjunto de librerías y paquetes con funcionalidades para cálculo, estadística, gráficas y aprendizaje automático, entre otras.

Scikit learn: es una librería para Python, que partiendo de las funcionalidades ofrecidas por otras librerias como NumPy, SciPy y Matplotlib, ofrece utilidades que permiten ejecutar de forma sencilla algoritmos de aprendizaje automático y modelado estadístico, incluyendo regresión, clasificación y clustering.

Tensor Flow: Es un framework para aprendizaje profundo desarrollado por Google y se utiliza con una API de Python. Es de código abierto y utiliza el concepto de grafos de computación donde los datos, en forma de tensores fluyen entre distintas operaciones. Tensor Flow puede utilizarse para una gran cantidad de aplicaciones que precisen de

cálculos numéricos, aunque su uso más popular es para aplicaciones de aprendizaje automático y, especialmente para aprendizaje profundo.

Keras: Es una librería de alto nivel con una interfaz modular en Python para el desarrollo de redes neuronales. Facilita la tarea a gran parte de los desarrolladores que no necesitan definir arquitecturas de bajo nivel. Internamente, Keras funciona sobre TensorFlow, aunque también es posible usar como backend otros frameworks como Theano y CNTK.

2.3 Aplicación de aprendizaje automático en el diagnóstico de enfermedades

Uno de los campos de aplicación de la inteligencia artificial en la actualidad es el sector salud. Los algoritmos de aprendizaje automático han sido ampliamente usados en tareas de predicción y clasificación de enfermedades. Existen investigaciones como la de Yu, Liu, Valdez, Gwinn y Khoury (2010), quienes usaron un algoritmo de Support Vector Machine (SVM) para predecir y clasificar pacientes con diabetes, utilizando como datos de entrada las variables de edad, etnia, peso, talla, antecedentes familiares y otros datos clínicos del paciente. El estudio logró un valor ROC del 83.5%.

De manera similar, Magnin et al. (2009) construyeron un modelo basado en SVM para clasificar un conjunto de 38 pacientes, e identificar entre aquellos que tienen Alzheimer y aquellos pertenecientes al grupo de control compuesto por personas de edad avanzada. El algoritmo se entrenó empleando como datos de entrada imágenes del cerebro de los pacientes, capturadas usando técnicas de imagen por resonancia magnética (MRI). Los resultados del estudio arrojaron un porcentaje de clasificación correcta del 94.5%, demostrando la capacidad del algoritmo de SVM para tareas de clasificación.

Dessai (2013) desarrolló un sistema de predicción de enfermedades del corazón mediante una red neuronal probabilística. Esta red neuronal con función de base radial se entrenó utilizando un conjunto de datos de 500 pacientes que contenía 13 atributos médicos capturados de la base de datos de enfermedades del corazón de Cleveland. El modelo obtuvo un 92.10% de los casos clasificados de forma correcta.

Por otra parte, Zhou, Jiang, Yang y Chen (2002) crearon una red neuronal basada en técnicas de ensemble que permitía identificar pacientes con cáncer de pulmón a partir de imágenes tomadas a las células obtenidas mediante biopsia.

Otro ejemplo del uso de redes neuronales en detección de enfermedades es el de Cao, Hu, Liu, Deng y Hu (2013) quienes desarrollaron un clasificador basado en perceptrón multicapa para la predicción de la cirrosis hepática inducida por el virus de la Hepatitis B. Este sistema que buscaba ser una alternativa a una técnica para la detección de cirrosis hepática cuyo procedimiento es altamente invasivo, logró entrenar un conjunto de datos provenientes de 124 pacientes con cirrosis hepática y 115 personas sin esta patología, obteniendo un valor de exactitud del 89.9%.

2.4 Redes neuronales y enfermedad renal crónica

En cuanto al pronóstico de la enfermedad renal crónica usando redes neuronales también existen estudios como los de Kumar y Abhishek (2009), quienes desarrollaron un algoritmo para la detección de cálculos renales tomando un conjunto de 1000 observaciones de pacientes, divididas en dos conjuntos, uno de personas que presentan cálculos renales y uno de personas sanas. Las entradas para el entrenamiento de la red son los resultados de diferentes exámenes de laboratorio, entre estos: prueba de creatinina, ácido úrico, glucosa, linfocitos y otros componentes sanguíneos. Se realizó el entrenamiento y pruebas con tres topologías de red neuronal diferente, obteniendo valores de exactitud de 92%, 87% y 84%, siendo la topología de perceptrón con dos capas ocultas y algoritmo de backpropagation la que obtuvo una mayor precisión.

El trabajo de Zhang, Hung, Chu, Chiu y Tang (2018) se enfocó en la aplicación de redes neuronales para predecir la supervivencia de pacientes con ERC. Esta es una tarea compleja debido a la cantidad de variables a tener en cuenta para determinar el tiempo de vida restante en personas que se encuentran en una etapa avanzada de la enfermedad. La evaluación precisa de la condición de los pacientes es de gran importancia, ya que permite identificar la atención adecuada, los medicamentos o las intervenciones médicas necesarias en cada caso. En esta investigación, el preprocesamiento de datos, las transformaciones de datos y las redes neuronales artificiales se utilizan para establecer el mapeo de muchos factores clínicos y su relación con la supervivencia del paciente.

En otros estudios se propone un modelo de redes neuronales entrenado con un algoritmo genético (GA) para detectar la enfermedad renal crónica (Hore, Chatterjee, Shaw, Dey y Virmani, 2018). La matriz de pesos de entrada de la red neuronal se optimiza gradualmente usando un algoritmo genético. El modelo se ha comparado con clasificadores conocidos como random forest y con otros modelos de redes neuronales como el perceptrón multicapa.

Los resultados experimentales sugieren que el modelo basado en redes neuronales con algoritmo genérico es capaz de detectar la ERC de manera más eficiente que los demás modelos.

La investigación de Al Imran, Amin y Johora (2018) aplicó tres técnicas de aprendizaje automático: regresión logística, redes neuronales tradicionales y redes neuronales profundas para diagnosticar la ERC. Para evaluar su rendimiento, se utilizaron métricas de F1, precisión, recall y AUC. El estudio concluyó que la red neuronal es la mejor técnica para el diagnóstico de ERC con un valor de F1 igual a 0.99, una precisión de 0.97, un recall de 0.99 y un AUC de 0.99. La regresión logística produjo el resultado más bajo entre todos y el modelo de aprendizaje profundo produjo buenos resultados en conjuntos de datos más grandes.

Otra investigación relacionada con aprendizaje profundo fue desarrollada por Kriplani, Patel y Roy (2019). En este caso se empleó un conjunto de datos de 224 registros de pacientes con enfermedad renal crónica. El método aplicado se basa en una red neuronal profunda que predice la presencia o ausencia de enfermedad renal crónica con una precisión del 97%. En comparación con otros algoritmos disponibles, el modelo desarrollado muestra mejores resultados, que se implementa mediante la técnica de validación cruzada para evitar que el modelo se adapte en exceso y presente sobreajuste.

Otra investigación relacionada con ERC es la de Rady y Anwar (2019), quienes construyeron una red neuronal para clasificar la etapa de la enfermedad renal crónica en la que se encuentra el paciente, a partir de 24 variables tomadas de su información clínica, como lo son sexo, edad, antecedentes de enfermedades como hipertensión y diabetes, y resultados de exámenes de laboratorio. En este estudio el conjunto de datos correspondía a 361 personas que ya habían sido diagnosticadas con ERC. Se utilizaron 3 algoritmos de redes neuronales y 1 algoritmo de SVM, obteniendo un valor de precisión del 96.7% para la Red neuronal probabilística, de 87.0% para la red neuronal con función de base radial, de 60.7% para la máquina de soporte vectorial, por sus siglas en inglés *Support Vector Machine* (SVM), y de 51.5% para la red neuronal con perceptrón multicapa. En este último estudio se concluyó que el algoritmo que clasificó mejor los datos fue la red neuronal probabilística, que consiste en una red neuronal cuya función de activación es de base radial y tiene un algoritmo de aprendizaje con una estructura altamente paralela.

También Xiao et al. (2019) realizaron una investigación empleando diferentes técnicas de aprendizaje automático para identificar el grado de progresión en pacientes con enfermedad renal crónica. Se recopiló información clínica de 551 pacientes con ERC y se seleccionaron

13 variables relacionadas con los resultados de pruebas de laboratorio junto con 5 variables de tipo demográfico.

El trabajo doctoral de Soldevila (2017) también busca predecir la evolución de la enfermedad renal usando técnicas de aprendizaje automático. Para este estudio los modelos se entrenaron con variables clínicas y analíticas, a partir de un conjunto de datos de 115 pacientes diagnosticados con ERC y tratados en la Unidad de Enfermedad Renal Crónica del Hospital La Fe de Valencia.

Un estudio que se acerca bastante al objetivo del presente trabajo es el de Ren, Fei, Liang, Ji y Cheng (2019), quienes desarrollaron una red neuronal híbrida con el fin de pronosticar ERC en pacientes con hipertensión arterial a partir de sus registros médicos. Los resultados del experimento demostraron que la red neuronal fue capaz de predecir correctamente los casos de ERC con una exactitud del 89.7%.

En todos estos estudios se utiliza como variables para el entrenamiento de los modelos la información de pruebas de laboratorio y se dispone de un conjunto relativamente pequeño de pacientes. En el caso del presente trabajo se cuenta con un gran conjunto de datos, que incluye todas las personas en Colombia que han tenido una atención en salud en el periodo de tiempo comprendido entre los años 2009 y 2018, es decir un volumen de 2.592.712.480 registros de atenciones correspondientes a 60.689.701 personas únicas, de acuerdo con las cifras oficiales del gobierno de Colombia (Ministerio de Salud y Protección Social, 2019). Estas variables no incluyen información de resultados de pruebas diagnósticos, pero si incluye información demográfica, como sexo, edad, etnia y lugar de residencia, así como el historial de las patologías que le han sido diagnosticadas a la persona, codificadas de acuerdo a la Clasificación Internacional de Enfermedades (CIE) en su versión 10.

Capítulo 3 – Desarrollo del software

En este capítulo se describen las técnicas empleadas siguiendo la metodología CRISP-DM con el fin de realizar la captura y de tratamiento de los datos, entrenamiento y evaluación del modelo de aprendizaje automático con el propósito de construir una red neuronal que permita pronosticar el riesgo de desarrollar enfermedad renal crónica a partir de los registros médicos de la población colombiana.

3.1 Metodología

Para el desarrollo del trabajo se empleará la metodología CRISP-DM, considerada un estándar para el ciclo de vida de proyectos de analítica de datos. CRISP-DM, siglas de *Cross-Industry Standard Process for Data Mining*, fue creado inicialmente para orientar proyectos en el ámbito de la minería de datos. Esta metodología incluye descripciones de las fases básicas de un proyecto, las tareas necesarias en cada fase y las relaciones entre las tareas. Entre sus ventajas frente a otras metodologías se encuentra su flexibilidad, ya que se puede adaptar fácilmente a cualquier proyecto de explotación de datos, como es el caso del aprendizaje automático (IBM Corporation, 2012).

De acuerdo con CRISP-DM, el ciclo de vida de vida del proyecto consiste en las seis fases mostradas en la siguiente figura.

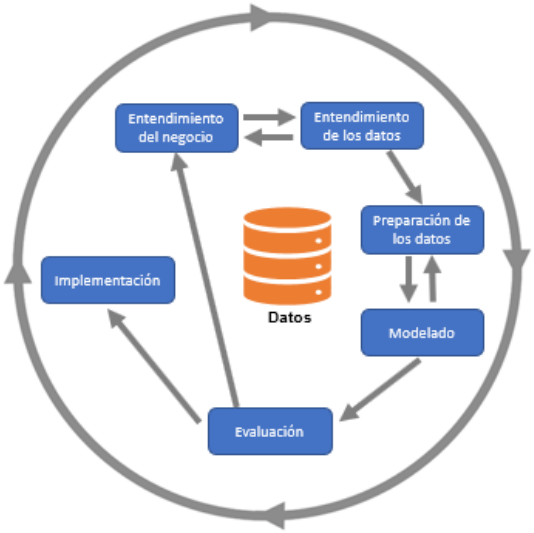


Figura 6. Metodología CRISP-DM.
Fuente: Elaboración propia basada en Chapman et al. (2000)

La secuencia de las fases no es rígida y se permiten movimientos hacia atrás y hacia adelante entre las diferentes etapas. Las flechas indican las secuencias más frecuentes de una fase a otra. El círculo externo en la figura simboliza la naturaleza cíclica de los proyectos de análisis de datos en donde este no se termina una vez que la solución se implementa, sino que la información descubierta durante el proyecto puede generar nuevas iteraciones del modelo (Chapman et al., 2000). A continuación, se describe brevemente cada fase del modelo:

1. **Comprensión del negocio**. Esta fase inicial se enfoca en la comprensión de los objetivos de proyecto. Después se convierte este conocimiento de los datos en la definición de un problema de análisis de datos y en un plan preliminar diseñado para alcanzar los objetivos.

2. **Entendimiento de datos**. La fase de entendimiento de datos comienza con la colección de datos inicial y continúa con las actividades que permiten familiarizarse con los datos, identificar los problemas de calidad, descubrir conocimiento preliminar sobre los datos, y/o descubrir subconjuntos interesantes para formar hipótesis en cuanto a la información oculta.

3. **Preparación de los datos**. La fase de preparación de datos cubre todas las actividades necesarias para construir el conjunto final de datos (los datos que se utilizarán en las herramientas de modelado) a partir de los datos en bruto iniciales. Las tareas incluyen la selección de tablas, registros y atributos, así como la transformación y la limpieza de datos para las herramientas que modelan.

4. **Creación del modelo**. En esta fase, se seleccionan y aplican las técnicas de modelado que sean pertinentes al problema, y se calibran sus parámetros a valores óptimos. En algunas ocasiones es necesario volver a la fase de preparación para realizar nuevas transformaciones sobre los datos.

5. **Evaluación del modelo**. En esta etapa en el proyecto, se han construido uno o varios modelos que parecen alcanzar calidad suficiente desde la una perspectiva de análisis de datos. Antes de proceder al despliegue final del modelo, es importante evaluarlo a fondo y revisar los pasos ejecutados para crearlo, comparar el modelo obtenido con los objetivos de negocio.

6. **Implementación**. Finalmente se realiza la puesta en producción del modelo. Dependiendo de los requisitos la fase de desarrollo puede ser tan simple como la generación de un informe o tan compleja como la realización periódica y quizás automatizada de un proceso de análisis de datos en la organización.

3.2 Comprensión del negocio

El ministerio de salud y protección social es el órgano rector del sector salud en Colombia. Su misión es “dirigir el sistema de salud a través de políticas de promoción de la salud, la prevención, el tratamiento y la rehabilitación de la enfermedad, con el fin de contribuir al mejoramiento de la salud de los habitantes de Colombia” (Ministerio de Salud y Protección Social, 2019). En el marco de esta misión, el ministerio realiza investigación epidemiológica para conocer el estado de salud de sus habitantes y la presencia de enfermedades en las diferentes poblaciones. Esto con el fin de anticipar el avance de las enfermedades y diseñar y ejecutar políticas públicas enfocadas en la promoción y prevención de estas.

Las tecnologías de la información y la comunicación han facilitado las actividades de recolección, tratamiento y análisis de los datos del sector salud. Los sistemas de información que se han creado permiten monitorear de forma permanente la presencia de enfermedades en todo el territorio nacional. Uno de los sistemas de información más importantes que se han desarrollado con este objetivo es el Registro Individual de Prestación de Servicios de Salud (RIPS). El RIPS se creó mediante la Resolución 3374 de 2000 emitida por el ministerio de salud y protección social, por la cual “se reglamentan los datos básicos que deben reportar los prestadores de servicios de salud y las entidades administradoras de planes de beneficios sobre los servicios de salud prestados” (Ministerio de Salud y Protección Social, 2019).

La norma establece que los prestadores de servicios de salud en Colombia deben reportar al ministerio los registros correspondientes a las atenciones médicas realizadas a los usuarios del sistema de salud. Los datos de este registro hacen referencia a la identificación del prestador del servicio, el usuario que lo recibe, la prestación del servicio propiamente dicho y el motivo que originó su prestación, es decir el diagnóstico dado al paciente (Ministerio de Salud y Protección Social, 2019). Esta información es altamente confiable teniendo en cuenta que su reporte se soporta en la expedición de la factura de venta del servicio. La factura es expedida por las instituciones prestadoras, quienes pueden ser: hospitales, clínicas, centros de salud o profesionales que de forma independiente ofrezcan servicios de salud. Esta factura funciona también como medio de control y de asignación de recursos financieros, ya que una pequeña parte del servicio es costeada por el usuario y el valor restante es pagado por el sistema de salud. Se debe tener en cuenta que las atenciones reportadas son aquellas que se brindan en el marco del Plan Obligatorio de Salud (POS), es decir de los servicios obligatorios que el sistema de salud debe ofrecer a sus usuarios. Las fuentes de datos de estos registros son las facturas de venta de servicios y las historias clínicas de los pacientes.

Existen otros sistemas de información y bases de datos que permiten obtener información sobre servicios de salud y patologías específicas, como lo son:

1. SIVIGILA: Eventos de interés en salud pública, como sarampión, fiebre amarilla, sika y demás.
2. PEDT: Información relacionada con actividades de protección específica y detección temprana.
3. BDUA: Base de datos única de afiliados al sistema de salud.
4. RUAF-ND: Sistema de información en línea sobre nacimientos y defunciones.
5. RLCPD: Registro de localización y caracterización de personas en condición de discapacidad.

Bases de datos de la Cuenta de Alto Costo: recopilan información sobre patologías de alto costo como VIH, cáncer, hemofilia, enfermedades huérfanas, artritis, enfermedad renal crónica, hipertensión arterial y diabetes mellitus.

Esta última base de datos contiene información detallada sobre los pacientes con enfermedad renal crónica, sin embargo, para el presente trabajo se eligió trabajar con la base de datos de RIPS debido a que contiene un mayor conjunto de registros, correspondientes a la totalidad de patologías reportadas para todos usuarios del sistema de salud.

3.2.1 Base de datos RIPS

Los datos que se reportan en el Registro Individual de Prestación de Servicios de Salud (RIPS) son los siguientes:

Datos del usuario: corresponde a los datos de identificación del paciente, sexo, edad, ocupación, lugar de residencia e información relacionada a su afiliación al sistema de salud.

Datos de consultas: son las atenciones en salud que recibe el paciente aplicable a todo tipo de consulta, programada o de urgencia, médica general y especializada, odontológica general y especializada y las realizadas por otros profesionales de la salud. Incluye información del centro médico que presta el servicio de consulta, la fecha, finalidad de la consulta, diagnóstico (de acuerdo con la clasificación internacional de enfermedades CIE) y los costos del servicio prestado.

Datos de procedimientos: corresponde a las actividades realizadas a los usuarios del sistema de salud, como pueden ser cirugías, exámenes u otros procedimientos diagnósticos o terapéuticos, de detección temprana o de protección específica. Incluye información del

prestador que realiza el procedimiento, la fecha, código del procedimiento (de acuerdo con la clasificación CUPS expedida por el ministerio de salud), diagnóstico (de acuerdo con la clasificación internacional de enfermedades CIE) y los costos del servicio prestado.

Datos de hospitalización: son las atenciones en salud que se prestan en un centro médico cuando el paciente requiere una atención permanente por parte del personal sanitario en un periodo de tiempo determinado. Incluye información del prestador, la fecha de ingreso y de salida, causa de ingreso, diagnóstico (de acuerdo con la clasificación internacional de enfermedades CIE), el estado de salida del paciente y su causa de muerte (si llegara a ocurrir).

Datos de urgencias: corresponden a la prestación de servicios de salud de urgencias, incluye las consultas y procedimientos realizados durante la estancia del paciente en este servicio. Los datos incluidos en este archivo son iguales a los reportados en los datos de hospitalización.

Datos de recién nacidos: es la información relacionada con el hecho vital del nacimiento. Incluye la fecha y hora, el sexo del recién nacido, peso, edad gestacional y diagnóstico de muerte (si llegara a ocurrir).

Datos de medicamentos: están relacionados con la codificación, forma farmacológica, principios activos, cantidad y costo de los medicamentos suministrados a los usuarios del sistema de salud.

La *Figura 7* muestra un esquema de los archivos reportados al Registro Individual de Prestación de Servicios de Salud (RIPS).

Figura 7: Tipos de archivos reportados al RIPS.
Fuente: Elaboración propia

3.2.2 Uso de los datos de RIPS

El volumen total de atenciones en salud reportadas a través de RIPS para el periodo 2009-2018 es de 2.653.686.446, correspondientes a un total de 60.970.527 personas. En la *Tabla 1* se observa el número de registros y personas por cada año.

Tabla 1: Número de atenciones y personas reportadas en RIPS

Año	Número de personas	Número de Atenciones
2009	17.295.258	154.608.418
2010	18.637.483	165.867.132
2011	20.798.260	224.225.473
2012	22.421.101	269.217.008
2013	22.274.058	232.825.813
2014	26.294.918	317.599.578
2015	24.417.882	307.820.498
2016	22.192.529	215.416.391
2017	25.788.519	313.039.968
2018	28.948.084	424.623.946
Total	**60.970.527**	**2.653.686.446**

Fuente: Bodega de Datos del ministerio de salud y protección social (2019)

El RIPS permite realizar seguimiento y control a las entidades prestadoras de servicios de salud y a las administradoras de planes de beneficios, además de servir como soporte para regular y controlar la venta de servicios de salud. Adicional a esto, el RIPS permite llevar a cabo análisis epidemiológicos de la población colombiana, identificando las patologías que más afectan a cada grupo demográfico. Este último propósito es quizás uno de los que más interesa a los investigadores porque permite describir el comportamiento de las enfermedades teniendo en cuenta las características demográficas de los habitantes y formular acciones de promoción y prevención para evitar el aumento de la morbilidad y mortalidad en la población.

Actualmente, el ministerio de salud y protección social realiza el análisis de estos datos empleando técnicas de inteligencia de negocios. La entidad cuenta con un Data Warehouse que integra información depurada de las diferentes fuentes de información que tiene la organización, entre estas el RIPS. Esta información se procesa y dispone a los usuarios finales a través de cubos OLAP. Los cubos OLAP (OnLine Analytical Processing) contienen estadísticas precalculadas y agregadas de los datos, a las que los usuarios pueden acceder a través de herramientas de consulta y visualización de amplio uso como Excel, Tableau y Power BI. La *Figura 8* muestra una consulta realizada en el cubo OLAP de la fuente RIPS.

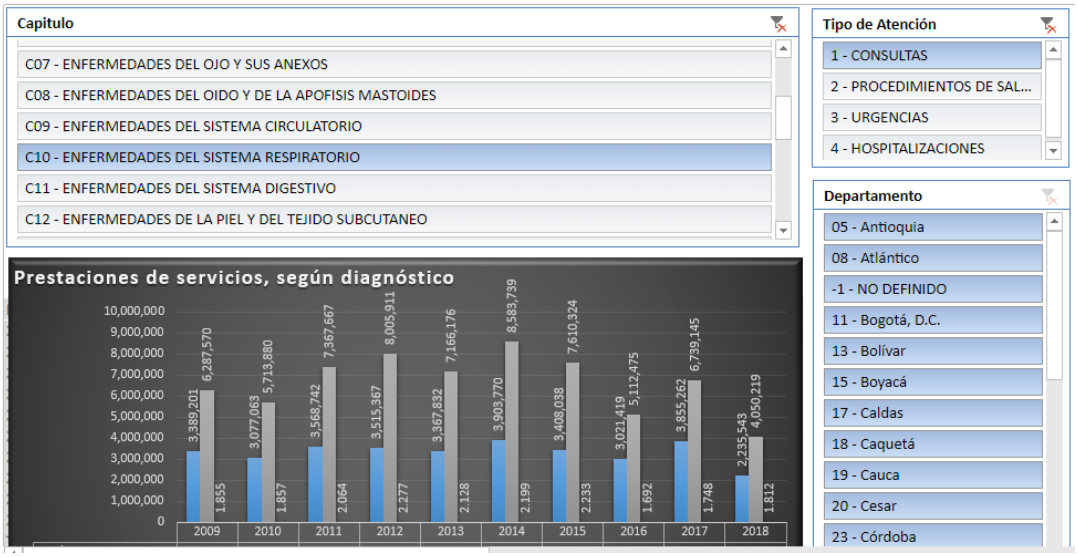


Figura 8: Cubo OLAP para la fuente RIPS.
Fuente: Bodega de Datos del ministerio de salud y protección social (2019)

A pesar de la utilidad de estos análisis, el ministerio desea utilizar la información proveniente de RIPS para realizar análisis predictivos, que permitan pronosticar el comportamiento futuro de las enfermedades y los riesgos en salud a los que se encuentran expuestos los usuarios del sistema, de acuerdo a su edad, sexo, etnia, ubicación geográfica y antecedentes médicos y familiares. A través del uso de técnicas de inteligencia artificial, y específicamente de aprendizaje automático, es posible procesar estos grandes volúmenes de datos con el fin de identificar patrones y realizar análisis predictivos.

Una de las aplicaciones de la inteligencia artificial en medicina es el diagnóstico de enfermedades a partir de conjuntos de datos obtenidos de historias médicas y pruebas diagnósticas. En el caso específico del ministerio, se requiere identificar las personas en riesgo de desarrollar enfermedades que generen altos costos al sistema, debido al precio de los medicamentos y procedimientos necesarios para su tratamiento. Estas patologías, como la enfermedad renal crónica (ERC) causan disminución en la calidad de vida del paciente y lo puede llevar incluso a la muerte.

Existen factores que pueden ayudar a identificar si una persona tiene un mayor riesgo de desarrollar ERC. Algunos de estos factores de riesgo están asociados a aspectos demográficos, como lo pueden ser: la edad de la persona, teniendo en cuenta que la ERC es más frecuente en adultos mayores, y la etnia, debido a que se presenta más en personas afrodescendientes. Otros factores demográficos como la geografía de residencia y el sexo también pueden llegar a tener cierta influencia. Por otra parte, existen numerosos estudios que demuestran que la presencia de enfermedades como hipertensión arterial, diabetes mellitus, enfermedades autoinmunes, afecciones sistémicas, infecciones del tracto urinario y

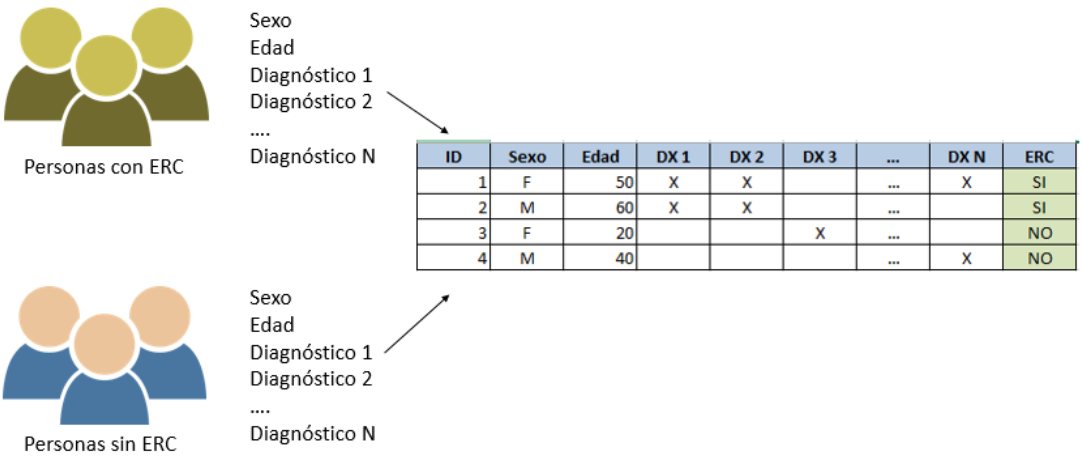
cálculos renales también pueden incidir directamente en el desarrollo de la ERC (Flores et al., 2009). De esta manera se plantea el objetivo de crear un modelo de aprendizaje automático que se entrene con un subconjunto de datos demográficos y de atenciones en salud tomados del RIPS, y genere como salida una predicción sobre el riesgo de desarrollar ERC por parte de los usuarios del sistema de salud en Colombia.

3.3 Captura y exploración de los datos

En esta sección se describe el conjunto de datos necesarios para el entrenamiento y pruebas del modelo de redes neuronales. También se realiza una exploración inicial de los datos para obtener conocimiento preliminar e identificar posibles problemas de calidad.

3.3.1 Conjuntos de datos

El conjunto de datos requerido para entrenar el modelo se obtuvo del sistema de información RIPS descrito en la sección anterior. Se tomaron dos muestras de la población: una correspondiente a personas con diagnóstico de ERC, y otra con el grupo de control de personas a las que no se les había diagnosticado la enfermedad. Para cada conjunto de datos se seleccionó el mismo grupo de variables correspondientes por una parte a datos demográficos de la persona y por otra a los diagnósticos identificados en el RIPS. Se adiciono una variable para identificar si la persona pertenecía al grupo de sujetos con ERC o si por el contrario pertenecía al grupo de control (Figura 9).



ID	Sexo	Edad	DX 1	DX 2	DX 3	...	DX N	ERC
1	F	50	X	X		...	X	SI
2	M	60	X	X		...		SI
3	F	20			X	...		NO
4	M	40				...	X	NO

Figura 9: Diseño del conjunto de datos requeridos para el modelo.
Fuente: Elaboración propia

Para obtener la muestra de individuos con ERC se identificaron las personas cuyos diagnósticos en la base de datos de RIPS correspondían con códigos dentro del subgrupo N18 de la Clasificación Internacional de Enfermedades (CIE). Este subgrupo corresponde a los diagnósticos de “Insuficiencia renal crónica”. Para cada persona se identificó la fecha de la primera atención con este diagnóstico y se etiquetó como “Fecha de diagnóstico”. Luego se seleccionaron aquellas personas cuya fecha de diagnóstico se encontraba comprendida entre el 01-01-2018 y el 31-12-2018. El número de personas obtenida en esta consulta fue de 203.015. Teniendo en cuenta que el entrenamiento del modelo puede llegar a ser costoso con un conjunto de datos tan grande se tomó una muestra aleatoria del 10% de las personas.

Para obtener el conjunto de personas del grupo de control se identificaron aquellas personas con atenciones en RIPS para el año 2018 y se tomó una muestra aleatoria con el mismo número de elementos existente en el grupo de personas con ERC, es decir 20.000 individuos. En este grupo de personas se realizó un filtro para descartar aquellos a los que se les había diagnosticado ERC en cualquier año.

Finalmente se integraron los dos grupos de personas en una misma tabla denominada “Personas_Unicas”, indicando para cada registro dos campos: el identificador único anonimizado de la persona y la etiqueta que indica con un “1” si la persona pertenece al grupo de ERC o con un “0” en caso contrario. El número de registros en esta tabla es de 40.000. En la *Figura 10* se muestra la tabla “Personas_Unicas”.

PersonaID	ERC
43547711	1
36745135	1
19575485	1
28055798	1
34640223	1
5031456	0
5775585	0
7527245	0
5321546	0
5033777	0

Figura 10: Tabla de personas únicas
Fuente: Elaboración propia

A partir de la tabla “Personas_Unicas” se generó otra tabla denominada “Personas”, en la que se agregaron las siguientes variables demográficas: sexo, edad, etnia y departamento de residencia de la persona. En la *Figura 11* se muestra el contenido de la tabla “Personas”.

PersonaID	Sexo	Edad	Etnia	Departamento	ERC
43547711	M	85	NINGUNA	BOGOTA_DC	1
36745135	M	72	NINGUNA	BOGOTA_DC	1
19575485	M	75	NINGUNA	RISARALDA	1
28055798	F	67	NINGUNA	VALLE_DEL_CAUCA	1
34640223	F	88	NINGUNA	BOGOTA_DC	1
5031456	F	63	INDIGENA	CORDOBA	0
5775585	F	38	INDIGENA	NARINO	0
7527245	F	54	NINGUNA	TOLIMA	0
5321546	F	35	NINGUNA	MAGDALENA	0
5033777	F	38	NINGUNA	BOGOTA_DC	0
5996558	F	51	NINGUNA	BOGOTA_DC	0
5069859	M	63	NINGUNA	CESAR	0
1024440	M	64	NINGUNA	VALLE_DEL_CAUCA	0
2545807	F	35	NINGUNA	BOGOTA_DC	0

Figura 11: Tabla de personas con variables demográficas
Fuente: Elaboración propia

Finalmente se creó la tabla “Diagnósticos” en la que se agregaron todos los diagnósticos identificados en el RIPS para cada persona de la tabla “Personas”, así como el número de atenciones médicas causadas por este diagnóstico, independientemente del tipo de atención, ya sean consultas, procedimientos, hospitalizaciones o urgencias. Se debe tener en cuenta que en esta consulta se retiraron los diagnósticos relacionados con el subgrupo N18. Esto con el fin de eliminar del conjunto de datos los diagnósticos de enfermedad renal crónica que tenían las personas del grupo de ERC. De otra forma durante el entrenamiento del modelo el algoritmo podría utilizar esta información para predecir de forma obvia el resultado, sin tener en cuenta los demás diagnósticos. El número de registros en esta tabla es de 1.064.588, correspondientes a 40.000 personas y 7.484 diagnósticos únicos. En la *Figura 12* se muestra el contenido de la tabla “Diagnósticos”.

PersonaID	CodigoDiagnostico	NumeroAtenciones
12575	J980	1
13677	K30X	4
18228	K802	2
18228	K805	2
18228	R101	2
18800	I872	2
20665	K297	1
21714	G990	1
75347	I255	1
125107	K296	1
127228	H526	1
140491	R103	3
300308	K000	2

Figura 12: Tabla de diagnósticos
Fuente: Elaboración propia

3.3.2 Exploración de datos

Una exploración previa de los datos permite identificar algunas características de cada variable. Entre estas características se puede identificar el tipo de variable (numérica o categórica), los valores mínimos y máximos para las variables numéricas, los valores de dominio para las variables categóricas y la existencia de valores faltantes o vacíos. En esta etapa es posible obtener una medida de la calidad de cada variable y la viabilidad para incluirla en el conjunto de datos final de entrenamiento del modelo de aprendizaje automático.

La primera tabla que se exploró fue la de "Personas". La variable sexo arrojó una distribución de 59% de mujeres y 41% de hombres en el conjunto de ERC; y una distribución del 53% de mujeres y 47% de hombres en el grupo de NO ERC. Este resultado muestra una mayor presencia de ERC en mujeres a diferencia de una distribución un poco más uniforme en el grupo de control. La *Figura 13* muestra la distribución de personas por sexo para cada conjunto de datos.

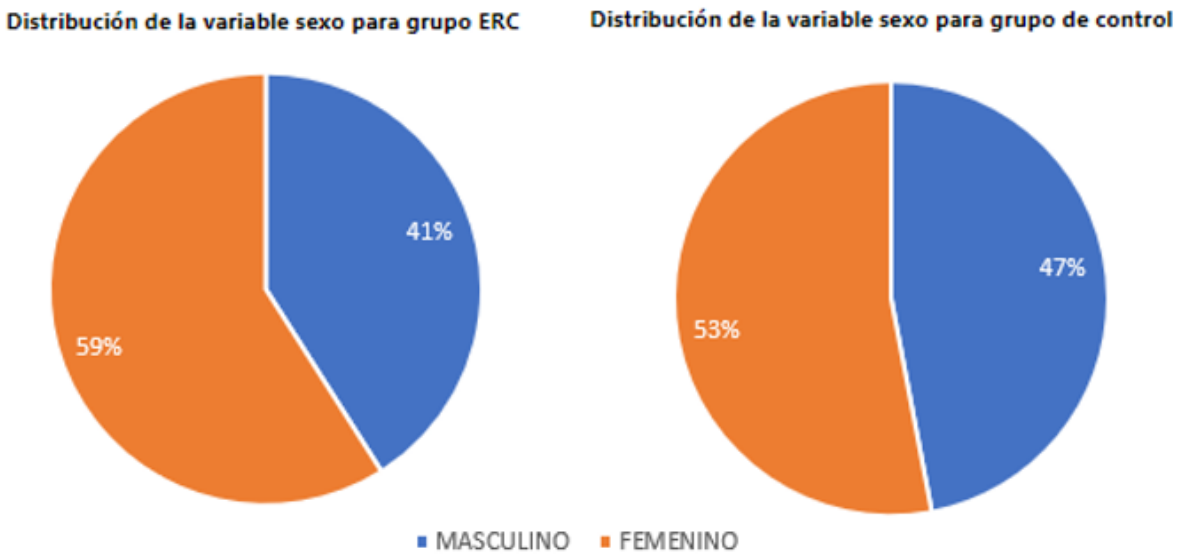


Figura 13: Distribución de personas por sexo
Fuente: Elaboración propia

El análisis de la variable etnia permite identificar que la mayor parte de la población colombiana no pertenece a ninguna etnia en particular. Sin embargo, se pueden observar algunos casos de personas pertenecientes a grupos indígenas, afrodescendientes y de otras etnias. La *Figura 14* muestra la distribución de personas por etnia para cada conjunto de datos.

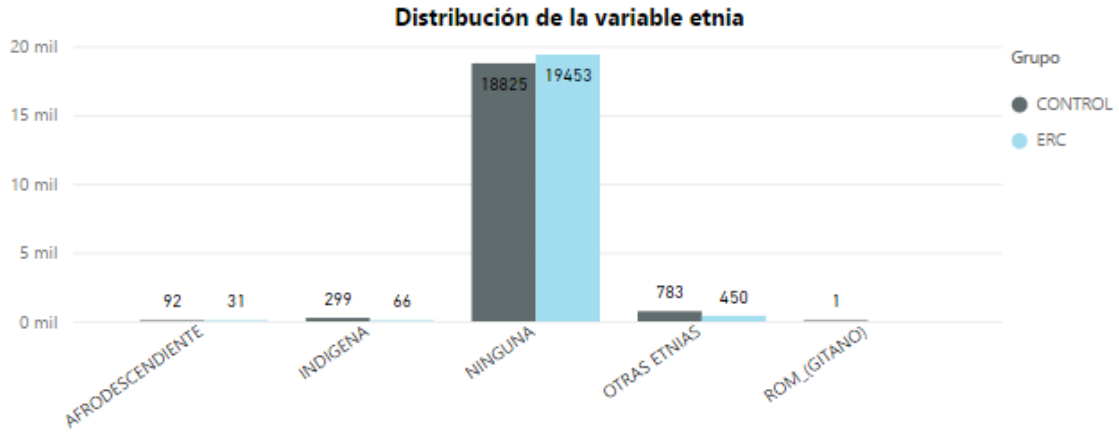


Figura 14: Distribución de personas por etnia
Fuente: Elaboración propia

La variable edad muestra que la ERC se presenta principalmente en personas mayores de 50 años, mientras que el grupo de control muestra una distribución similar para cada grupo de edad, aunque un poco más fuerte en menores de 60 años, teniendo un declive a medida que aumenta la edad y disminuye la población para estos grupos de edad. La *Figura 15* muestra la distribución de personas por edad para cada conjunto de datos.

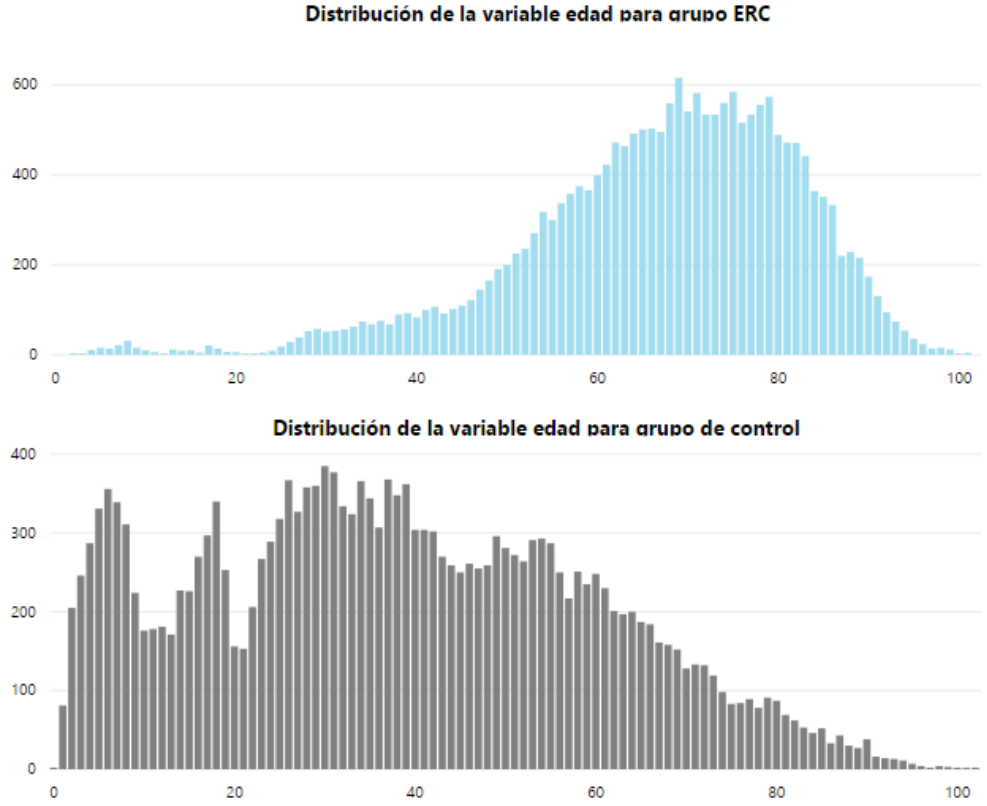


Figura 15: Distribución de personas por edad
Fuente: Elaboración propia

La última variable analizada para la tabla “Personas” es el departamento de residencia del paciente. Aquí se puede ver que los departamentos donde se presentan más casos de ERC, comparados con el grupo de control, son Bogotá, Valle del Cauca, Atlántico y Santander. La *Figura 16* muestra la distribución de personas por departamento para cada conjunto de datos.

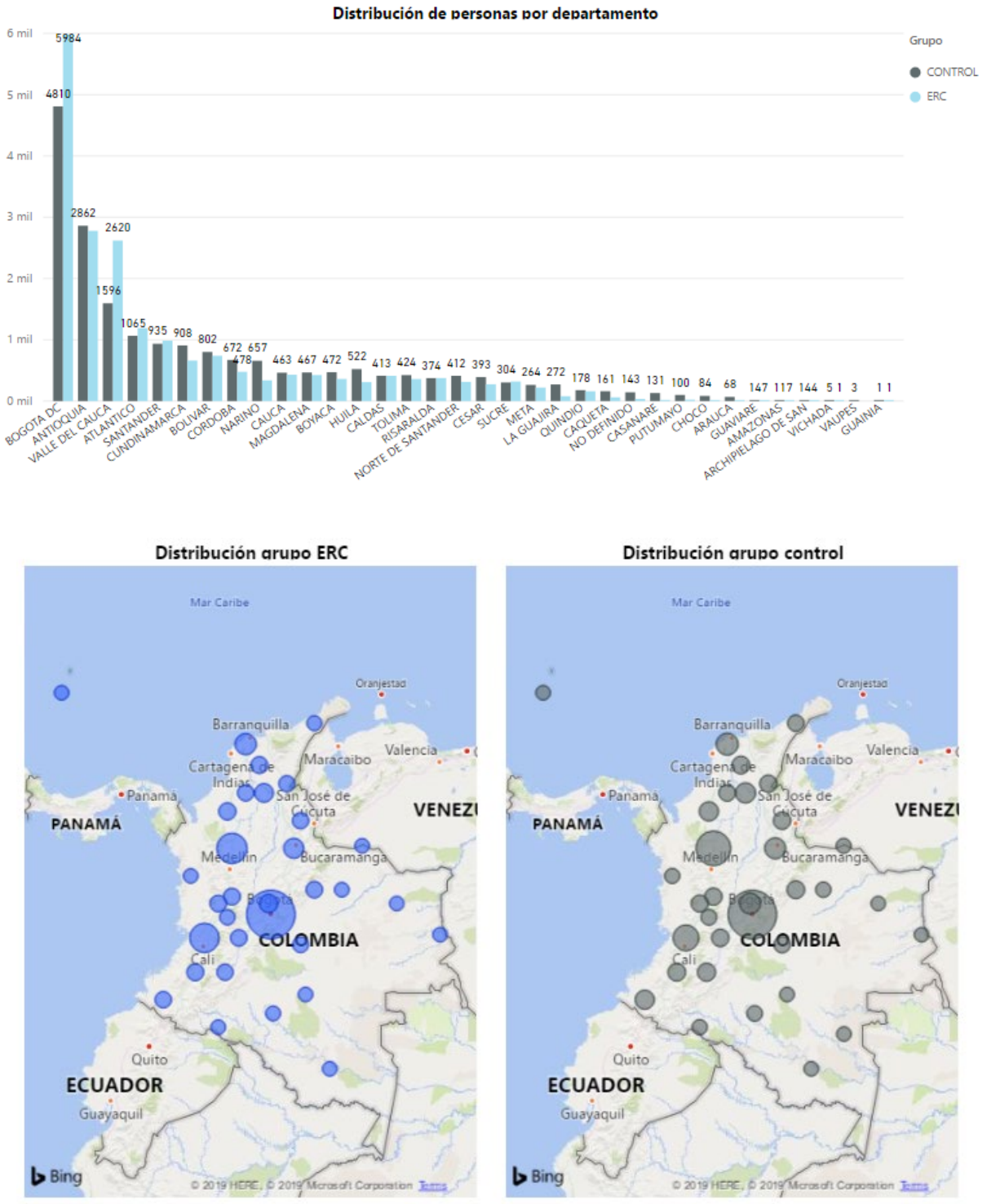


Figura 16. Distribución de personas por departamento
Fuente: Elaboración propia

Para la tabla “Atenciones” se analizó la variable Diagnóstico, que se encuentra codificada de acuerdo a la Clasificación internacional de enfermedades (CIE). Este código contiene cuatro dígitos que permiten identificar el capítulo, grupo, subgrupo y diagnóstico de cada enfermedad. La siguiente figura muestra los 10 diagnósticos más frecuentes por cada grupo. Es posible identificar que en el grupo de ERC los diagnósticos más frecuentes son hipertensión arterial, diabetes mellitus e infección de vías urinarias. Esto coincide con las teorías médicas que afirman que estas patologías son factores de riesgo de la ERC (Flores et al., 2009).

Primeros diez diagnósticos para el grupo ERC

Diagnostico	Numero de atenciones médicas
I10X - HIPERTENSION ESENCIAL (PRIMARIA)	550491
E119 - DIABETES MELLITUS NO INSULINODEPENDIENTE SIN MENCION D...	99961
K021 - CARIES DE LA DENTINA	83847
N390 - INFECCION DE VIAS URINARIAS, SITIO NO ESPECIFICADO	72474
E109 - DIABETES MELLITUS INSULINODEPENDIENTE SIN MENCION DE CO...	60093
M545 - LUMBAGO NO ESPECIFICADO	58976
R104 - OTROS DOLORES ABDOMINALES Y LOS NO ESPECIFICADOS	46169
E039 - HIPOTIROIDISMO, NO ESPECIFICADO	40662
J449 - ENFERMEDAD PULMONAR OBSTRUCTIVA CRONICA, NO ESPECIFIC...	37780
M255 - DOLOR EN ARTICULACION	33251

Primeros diez diagnósticos para el grupo de control

Diagnostico	Numero de atenciones médicas
K050 - GINGIVITIS AGUDA	6997
E669 - OBESIDAD, NO ESPECIFICADA	6957
R509 - FIEBRE, NO ESPECIFICADA	6833
R101 - DOLOR ABDOMINAL LOCALIZADO EN PARTE SUPERIOR	5909
J449 - ENFERMEDAD PULMONAR OBSTRUCTIVA CRONICA, NO ESPECIFIC...	5659
B24X - ENFERMEDAD POR VIRUS DE LA INMUNODEFICIENCIA HUMANA (...	5599
R102 - DOLOR PELVICO Y PERINEAL	5495
E782 - HIPERLIPIDEMIA MIXTA	5480
N939 - HEMORRAGIA VAGINAL Y UTERINA ANORMAL, NO ESPECIFICADA	5277
R520 - DOLOR AGUDO	5233

*Figura 17.*Diagnósticos más frecuentes para cada grupo.
Fuente: Elaboración propia

3.4 Limpieza y preparación de los datos

Este proceso transforma el conjunto de datos obtenido de la base de datos y genera un nuevo modelo con el que se va a entrenar y probar la red neuronal. Esta transformación incluye operaciones de selección de características, unión de tablas, transformación de filas a columnas, manejo de variables categóricas, tratamiento de valores nulos y otras operaciones de limpieza requeridas para obtener el conjunto de datos final.

3.4.1 Selección de características

A partir de la exploración inicial de los datos, se identificaron las siguientes variables con las que se va a entrenar el modelo:

Sexo: variable categórica con dos valores de dominio: “F” (Femenino) y “M” (Masculino).

Edad: variable numérica con valores enteros entre 0 y 130, que almacena la edad del paciente en la fecha de corte de los datos, es decir al 31 de diciembre de 2018.

Etnia: variable categórica con 4 valores de dominio: “INDIGENA”, “AFROCOLOMBIANO”, “OTRAS ETNIAS” y “NINGUNA”. El valor NINGUNA indica que la persona no se reconoce dentro ningún grupo étnico específico.

Departamento: variable categórica con 32 valores de dominio correspondientes a los nombres de los departamentos de Colombia. Representa la ubicación geográfica del paciente en la fecha de corte de los datos.

Diagnósticos: variable categórica con los códigos CIE de las 7.484 enfermedades identificadas en la tabla “Diagnósticos”. Para cada diagnóstico se almacena el número de atenciones reportadas en el RIPS por cada persona. Si la persona no tiene el diagnóstico el número de atenciones será igual a 0.

ERC: variable numérica que contiene dos valores: 1 si la persona tiene ERC y 0 en caso contrario. Esta variable se utilizará como etiqueta de cada elemento del conjunto de datos.

Existen otras variables que se consideraron para conformar el conjunto de datos, sin embargo, no se incluyeron debido a la inexistencia de datos, como en el caso de los resultados de pruebas de laboratorio y antecedentes familiares, o debido a problemas de calidad de los datos, como en el caso de las variables relacionadas con el peso y talla de las personas. Estas últimas variables se pueden obtener de la base de datos de PEDT, sin embargo, no cuentan con información disponible para la totalidad de la población.

3.4.2 Transformación de filas a columnas

La tabla “Diagnósticos” almacena la información correspondiente a las enfermedades detectadas a cada paciente. Cada fila corresponde a un par paciente-enfermedad. Para la creación del modelo de datos requerido para el entrenamiento de la red neuronal es necesario tener toda la información del paciente en el mismo registro. Por esta razón es necesario realizar una transformación de filas a columnas para que todas las patologías que tenga un paciente se encuentren en la misma fila. La *Figura 18* ilustra la transformación requerida para la tabla “Diagnósticos”.

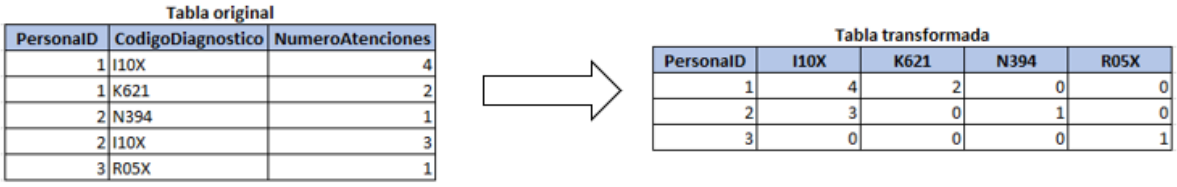


Tabla original

PersonalID	CodigoDiagnostico	NumeroAtenciones
1	I10X	4
1	K621	2
2	N394	1
2	I10X	3
3	R05X	1

Tabla transformada

PersonalID	I10X	K621	N394	R05X
1	4	2	0	0
2	3	0	1	0
3	0	0	0	1

Figura 18. Transformación de filas a columnas.
Fuente: Elaboración propia

La transformación de filas a columnas se realizó haciendo uso del lenguaje de programación Python 3.7.0, en el entorno de desarrollo Jupyter Notebook 5.6.0, y se empleó la función *pivot* incluida en la librería *pandas*. Esta función recibe como parámetros el campo índice, que en este caso es el identificador anonimizado de la persona, el campo que contiene los valores a transformar en columnas, en este caso corresponde al código del diagnóstico, y el campo por el que se van a agregar los datos, que para este caso es el número de atenciones.

Una vez ejecutada la función se genera un *dataframe* de *pandas* que contiene un registro para cada una de las 40.000 personas de la muestra, con 7.484 columnas correspondientes a cada enfermedad posible dentro de la base de datos, además del identificador único de la persona. El paso final consiste en rellenar los valores vacíos con ceros. Esto ocurre en los casos donde la persona no tiene alguna de las enfermedades. La *Figura 19* ilustra el resultado del proceso.

```
#Conversión de filas a columnas
dfDiagnosticos = dfDiagnosticos.pivot(index='PersonaID', columns='CodigoDiagnostico', values='NumeroAtenciones')
dfDiagnosticos = dfDiagnosticos.fillna(0)
dfDiagnosticos.head()
```

CodigoDiagnostico	A000	A001	A009	A010	A011	A012	A013	A014	A020	A021	...	Y899	Y909	Y910	Y911	Y912	Y913	Y919	Y95X	Y96X	Y98X
PersonaID																					
167006	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
595290	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
747179	0.0	2.0	0.0	0.0	0.0	0.0	2.0	4.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1500321	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1664002	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1823619	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2217241	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figura 19. Resultado de la transformación de filas a columnas.
Fuente: Elaboración propia

3.4.3 Transformación de variables categóricas

Las variables categóricas contienen descripciones dentro de un conjunto de elementos finitos que no se pueden convertir en valores numéricos. Esto ocurre con las variables sexo, etnia y departamento de la tabla “Persona”. Para estos casos es necesario crear tantas columnas como posibles valores de dominio contiene cada variable y diligenciar con valores “1” o “0” dependiendo de la opción que aplique para cada persona. La *Figura 20* ilustra la transformación requerida para la variable categórica “Sexo”.

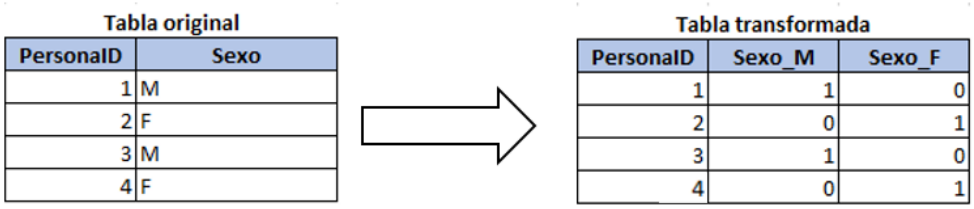


Figura 20. Transformación de variables categóricas.
Fuente: Elaboración propia

Esta transformación se realizó con la función *get_dummies* de la librería *pandas*, a la que se envía como parámetro el campo del *dataframe* que se va a dividir en varias columnas. A continuación, se muestra el resultado obtenido luego de aplicar la función a las variables Sexo, Etnia y Departamento. En el caso de la variable Etnia se retiró el valor “NINGUNA” y para la variable Departamento se retiró el valor “NO DEFINIDO”, teniendo en cuenta que no aportan información relevante para el entrenamiento del modelo.

```
#Transforma variables categóricas
dfPersonas = pd.get_dummies(dfPersonas, columns=['Sexo'], prefix = ['Sexo'])
dfPersonas = pd.get_dummies(dfPersonas, columns=['Etnia'], prefix = ['Etnia'])
dfPersonas = pd.get_dummies(dfPersonas, columns=['Departamento'], prefix = ['Departamento'])
dfPersonas = dfPersonas.drop(['Etnia_NINGUNA'], axis=1)
dfPersonas = dfPersonas.drop(['Departamento_NO_DEFINIDO'], axis=1)
dfPersonas.head()
```

	PersonalID	Edad	ERC	Sexo_F	Sexo_M	Etnia_INDIGENA	Etnia_NEGRO_MULATO_AFROC	Etnia_OTRAS_ETNIAS	Departamento_AMAZONAS
0	6839613	56	1	1	0	0	0	0	0
1	22295109	62	1	0	1	0	0	0	0
2	43524142	66	1	1	0	0	0	0	0
3	35157551	75	1	1	0	0	0	0	0
4	17735990	64	1	1	0	0	0	0	0

Figura 21. Resultado de la transformación de variables categórica.
Fuente: Elaboración propia

3.4.4 Unión de tablas

Un último paso requerido para la creación del conjunto de datos final consiste en realizar un merge o unión de las dos tablas trabajadas hasta el momento: la de “Personas” y la de “Diagnósticos”. Cada una de las tablas contiene un único registro por cada persona, de modo que el proceso consiste en unir la fila de la tabla “Personas” con la fila de la tabla “Diagnósticos” para la misma persona. Esta funcionalidad la aporta la función *merge* de la librería *pandas* en *Python*.

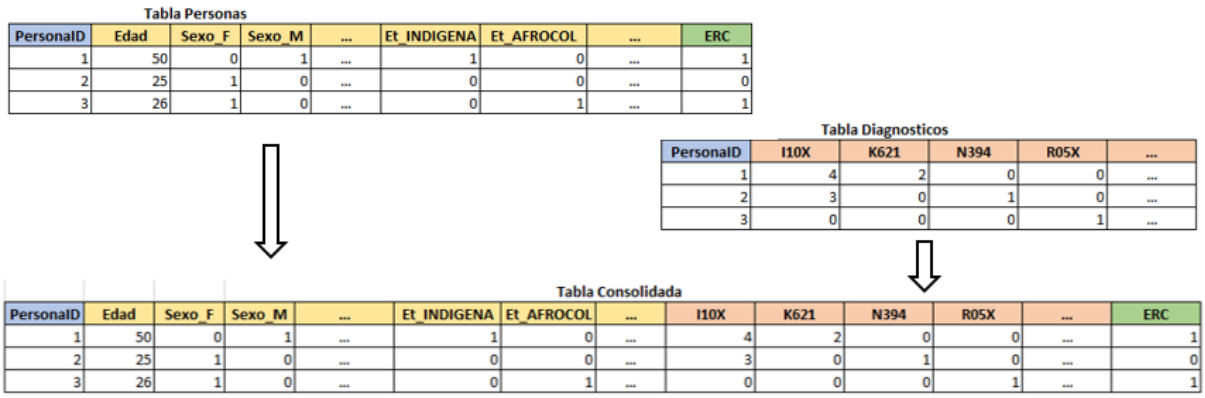


Figura 22. Unión de tablas de Personas y Diagnósticos
Fuente: Elaboración propia

A continuación, se presenta el conjunto final de datos, producto de la unión de las tablas de “Personas” y “Diagnósticos”.

```
#Realiza join entre los dos dataframes
dfConsolidado = pd.merge(dfDiagnosticos, dfPersonas, on='PersonaID', how='left')
dfConsolidado.head()
```

	PersonaID	A000	A001	A009	A010	A011	A012	A013	A014	A020	...	Departamento_NORTE_DE_SANTANDER	Departamento_PUTUMAYO
0	40	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0
1	60	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0
2	161	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0
3	1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0
4	1456	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0

Figura 23. Conjunto final de datos
Fuente: Elaboración propia

3.5 Modelado de la red neuronal

Las redes neuronales son un tipo de algoritmo de aprendizaje automático cuyo funcionamiento está inspirado en las conexiones de las neuronas del cerebro humano. Cada neurona artificial recibe como entrada la salida de otras neuronas y aplica una función matemática sobre los valores de entrada, para a su vez producir una señal de salida que se envía a otras neuronas que conforman la red. A medida que se agregan más capas de neuronas a la red, estas van aprendiendo conceptos más complejos a partir de los conceptos más simples aprendidos en las primeras capas de la red. Las redes neuronales han demostrado tener una gran capacidad para encontrar patrones entre las variables de entrada y de salida, especialmente cuando las relaciones entre estas no son lineales y sus patrones son complejos (Goodfellow, Bengio y Courville, 2016). El presente trabajo busca entrenar una red neuronal a partir los datos clínicos y demográficos de un conjunto de pacientes, con una variable de salida binaria que indica la presencia de enfermedad renal crónica.

3.5.1 Herramientas empleadas en el modelado de la red

Para la creación, evaluación e implementación de la red neuronal se utilizó el lenguaje de programación Python 3.7.0, en conjunto con la librería Keras 2.2.4 y el framework TensorFlow 1.13.1, sobre un entorno de desarrollo Jupyter Notebook 5.6.0. La elección de Python se basó en la popularidad de uso de dicho lenguaje, así como la existencia de múltiples librerías y frameworks, como Theano, PyTorch y Tensor Flow, que facilitan la creación de modelos basados en redes neuronales. Uno de los frameworks más utilizados para el entrenamiento de redes neuronales es TensorFlow, el cual fue desarrollado por Google para computación numérica y utiliza grafos de computación donde los datos fluyen entre distintas operaciones. Un grafo de computación es una representación de una serie de

operaciones matemáticas, donde los nodos se corresponden con operaciones y las aristas con datos (Abrahams, Hafner, Erwitt y Scarpinelli, 2016).

Las aplicaciones más comunes de TensorFlow están enfocadas en tareas de aprendizaje automático, y especialmente en aprendizaje profundo con redes neuronales. TensorFlow cuenta con una API para Python, aunque su núcleo esta desarrollado en C++, lo que le brinda una mayor velocidad de ejecución y un mejor aprovechamiento de los recursos computacionales. Una de las ventajas al utilizar Tensor Flow es la optimización de los cálculos a realizar, ya que las operaciones que no son realmente necesarias no son ejecutadas. Adicionalmente, si las operaciones a realizar son conocidas de antemano, se pueden realizar optimizaciones sobre ellas que repercutan en una mayor velocidad de ejecución. Por otra parte, TensorFlow facilita la ejecución distribuida en varias máquinas o GPU. De esta forma es posible dividir el grafo en varios subgrafos y hacer que distintos componentes del sistema distribuido se encarguen de diferentes partes (Abrahams et al., 2016).

TensorFlow es un framework muy potente y versátil, capaz de implementar a bajo nivel una gran variedad de redes neuronales. Sin embargo, en muchas ocasiones se requiere utilizar una arquitectura estándar que evite en gran medida tener que reescribir el código de los mismos elementos una y otra vez. Es por esto que TensorFlow dispone de librerías de alto nivel que permiten la definición de redes neuronales de una manera más sencilla y directa. Keras es tal vez la librería de alto nivel más utilizada. Esta librería tiene una interfaz modular en Python para el desarrollo de redes neuronales que facilita la tarea de los desarrolladores al no tener que definir arquitecturas de bajo nivel. Internamente, Keras funciona sobre TensorFlow, aunque también es posible usarlo sobre otros frameworks como Theano y CNTK (Gulli y Pal, 2017).

Finalmente, el entorno de desarrollo Jupyter Notebook facilita la creación y ejecución de código en Python a través de una interfaz web de fácil uso. Esta aplicación permite agregar bloques de código junto con bloques de texto con formato enriquecido que incluyen imágenes, gráficas, ecuaciones y HTML. Esto es muy útil para facilitar la documentación de los experimentos realizados con redes neuronales y algoritmos de aprendizaje automático.

3.5.2 Conjuntos de datos de entrenamiento, validación y pruebas

Una vez realizado el proceso de limpieza y preparación de los datos, estos se utilizarán para entrenar la red neuronal. En esta etapa del proceso es necesario separar el conjunto de datos en tres subconjuntos: entrenamiento, validación y pruebas. El conjunto de datos de entrenamiento está compuesto por aquellos ejemplos que permitirán al modelo aprender de las características e identificar los patrones ocultos en los datos. El conjunto de validación permite identificar si el modelo está aprendiendo correctamente a medida que se entrena. Esto es importante teniendo en cuenta que los modelos de aprendizaje automático pueden caer en sobreajuste, también denominado *overfitting* en inglés.

El sobreajuste consiste en que el modelo comienza a memorizar los datos con los que aprende, adaptándose a ellos y siendo incapaz de generalizar nuevos ejemplos. Una forma efectiva de identificar si un modelo presenta sobreajuste es utilizar un conjunto de datos de validación que permitan ir probando el desempeño del modelo a medida que se entrena. La *Figura 24* presenta un caso de sobreajuste en un conjunto de datos donde se puede apreciar como el error disminuye en el conjunto de datos de entrenamiento, pero aumenta en el en el conjunto de datos de validación. La línea punteada indica el momento en el que tanto el conjunto de entrenamiento como el de validación presentan su valor óptimo.

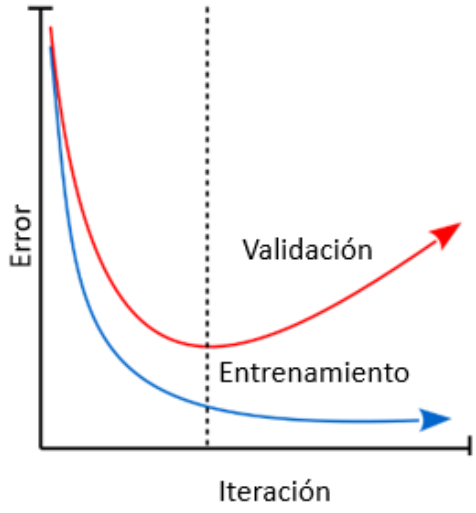


Figura 24. Ejemplo de sobreajuste.
Fuente: Elaboración propia.

El último grupo de datos, denominado conjunto de test o pruebas, consiste en una porción de ejemplos diferentes a los utilizados para entrenamiento y validación, usualmente entre el 20% y 30% del total de datos, que permiten comprobar el desempeño del modelo final obtenido. Los valores resultantes de aplicar las métricas de evaluación al conjunto de datos

de pruebas deben ser muy parecidos a los obtenidos con los otros dos conjuntos de datos. Valores muy alejados podrían indicar problemas durante el entrenamiento, como los causados por el sobreajuste, o fallos durante la separación aleatoria de los conjuntos que causan que los datos de pruebas tengan características diferentes a los empleados para entrenar el modelo.

Antes de proceder con la creación de los conjuntos de datos es necesario realizar tres operaciones adicionales. La primera de ellas consiste en una revisión de las variables, o *features* en inglés*,* empleadas para el entrenamiento del modelo. El conjunto de variables hasta este momento se encuentra compuesto por 7.494 variables obtenidas luego de aplicar procesos de limpieza y preparación de datos. Como resultado de esta revisión se identificó que la variable *PersonalID* no es relevante para el entrenamiento del modelo. Este campo fue útil para relacionar los datos demográficos y de diagnósticos previos de cada persona con el fin de conformar el conjunto de datos descrito en la *Figura 23*. Sin embargo, no es relevante para el entrenamiento de la red neuronal, ya que cada persona cuenta con un identificador diferente y la red no aprenderá nada de este campo. Una vez retirada la columna, el conjunto de datos resultante está compuesto por 40.000 registros y 7.493 variables.

La siguiente operación consiste en separar los datos en dos estructuras diferentes que dentro del lenguaje Python se conocen como dataframes. Por una parte, se tiene el dataframe *x*, que contiene las características o variables de entrada del modelo, y por otra parte se tiene el dataframe *y,* que contiene la clase o variable de salida del modelo. Para la creación del dataframe *y* se seleccionó únicamente la variable “ERC” que contiene la etiqueta que permite diferenciar aquellos pacientes que tienen ERC, de aquellos que no presentan la enfermedad. Por otra parte, el dataframe x se creó haciendo una copia del conjunto de datos original, y retirando luego la variable “ERC”. De esta manera el dataframe *x* contiene 40.000 registros con 7.492 variables, y el dataframe *y* contiene 40.000 registros con 1 variable binaria.

La tercera operación consiste en normalizar los datos. Cada variable del dataframe *x* contiene un rango de valores diferentes al de las demás variables o características. Esto puede ser un problema para el entrenamiento del modelo ya que las variables con valores más altos podrían llegar a tener mayor peso que las demás. Para resolver esta situación se realiza un proceso de normalización de los datos, de manera que todas las variables se encuentren en un rango de valores entre 0 y 1. La normalización de los datos se obtiene aplicando la siguiente operación matemática: x = (x - x.min()) / (x.max() - x.min()), donde a

cada dato se le resta el valor mínimo, y luego se divide por el valor resultante de la diferencia entre el valor máximo y el valor mínimo de los datos para cada variable.

Finalmente se realiza la separación de los conjuntos de datos de entrenamiento y pruebas haciendo uso de la función *train_test_split*, perteneciente a la librería *scikit-learn*. Esta función recibe como parámetros los dataframes que contienen las variables de entrada (x) y salida (y), junto como el porcentaje de los datos que se utilizarán para pruebas. En este caso se destinó el 30% de los datos para pruebas y el 70% restante de los ejemplos para entrenar el modelo. La función retorna cuatro dataframes que contienen los datos *x* y *y* para los conjuntos de entrenamiento y pruebas. El conjunto de datos de validación se obtendrá a partir del conjunto de entrenamiento durante la construcción de la red neuronal. La *Figura 25* muestra el llamado a la función *train_test_split* que realiza la separación de los datos.

```
#Separa datos de entrenamiento y pruebas
entrenamientoX, pruebasX, entrenamientoY, pruebasY = train_test_split(
    x,y, test_size= 0.30, random_state=2)
entrenamientoX.head()
```

Figura 25. Separación de los conjuntos de entrenamiento y pruebas.
Fuente: Elaboración propia en Python 3.7.0.

3.5.3 Topología de la red

La topología o arquitectura de la red define el número de capas, así como el número de neuronas por capa y la forma en que están conectadas. La topología de una red está directamente relacionada con la complejidad de las tareas que pueden ser aprendidas por esta (Goodfellow et al., 2016). Por lo general, redes con mayor número de capas y neuronas son capaces de identificar patrones más complejos, aunque consumen una mayor cantidad de recursos computacionales, especialmente capacidad de procesamiento y espacio en memoria.

La topología de una red neuronal está compuesta por lo general por una capa de entrada, una o más capas ocultas y una capa de salida. Las neuronas de la capa de entrada reciben directamente los datos de las características o variables del conjunto de datos. Por esta razón el número de neuronas de la capa de entrada es igual al número de variables del conjunto de datos. A continuación, la red puede tener varias capas de neuronas que toman los datos de las capas anteriores, las procesan y generan una única salida luego de aplicar una función matemática, llamada función de activación. La mayoría de las redes neuronales son totalmente conectadas lo cual implica que cada nodo en una capa se conecta a todos los nodos en la capa siguiente. La última capa de la red, denominada capa de salida,

contiene tantas neuronas como clases pueda tener el problema de clasificación. Para una clasificación binaria como la que se desea realizar en este caso, la red únicamente tiene una neurona que genera como salida la probabilidad de que el ejemplo pueda ser catalogado como positivo.

La red neuronal propuesta para el presente trabajo está compuesta por 5 capas de acuerdo con el diagrama de la *Figura 26*. La capa de entrada corresponde a las características o variables de entrada de la red, compuesta por 7.492 nodos o neuronas. Las siguientes 3 capas corresponden a las capas ocultas del modelo y contienen 500, 100 y 50 neuronas respectivamente. La última capa corresponde a la neurona que representa la única clase del problema de clasificación binario. El objetivo del entrenamiento es obtener los valores correspondientes a los pesos óptimos (W) para cada capa de la red, así como a los valores de *bias*.

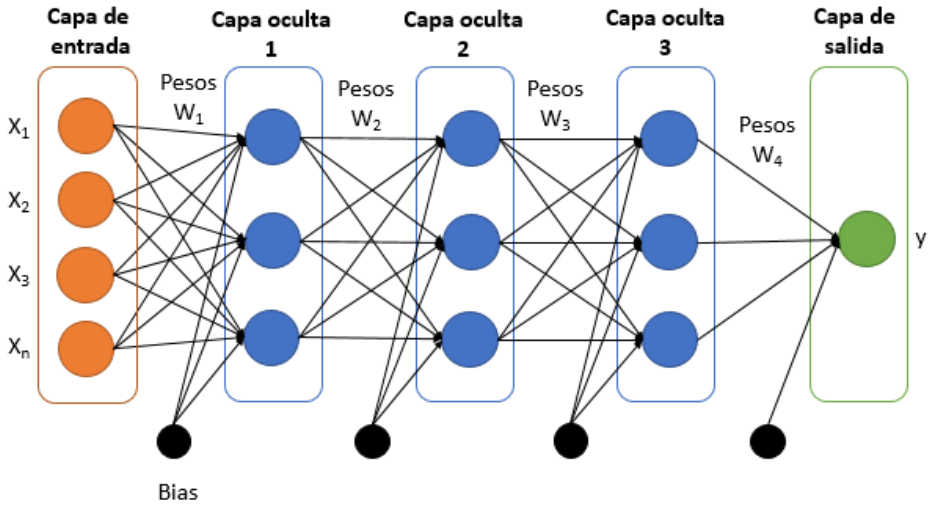


Figura 26. Topología de la red.
Fuente: Elaboración propia.

3.5.4 Función de activación

Uno de los elementos más críticos para el entrenamiento de redes neuronales es la función de activación. La función de activación es el mecanismo por el cual las neuronas artificiales procesan la información y esta se propaga por la red. La unidad de activación más tradicional es la de tipo sigmoide, utilizada de manera clásica en el mundo de las redes neuronales (Nielsen, 2015). Esta función se caracteriza por convertir cualquier número real en un número entre 0 y 1, haciendo que valores positivos grandes sean prácticamente 1, mientras que valores negativos grandes tiendan a 0. Es por eso que esta función se utiliza

normalmente para modelar probabilidades. La siguiente figura demuestra el comportamiento de la función sigmoide:

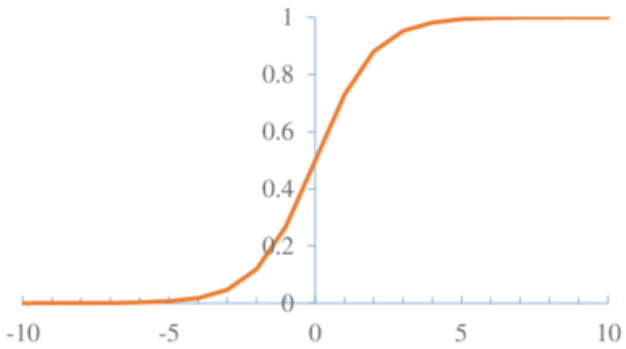


Figura 27. Función sigmoide. (Yang y Yang, 2018)

En la actualidad se utilizan otras funciones de activación que han resuelto varios de los problemas que presentaba la función sigmoide. Esto ha permitido entrenar redes cada vez más profundas y complejas. Una de estas funciones es la unidad *ReLU* (rectified linear unit) que tiene el siguiente comportamiento:

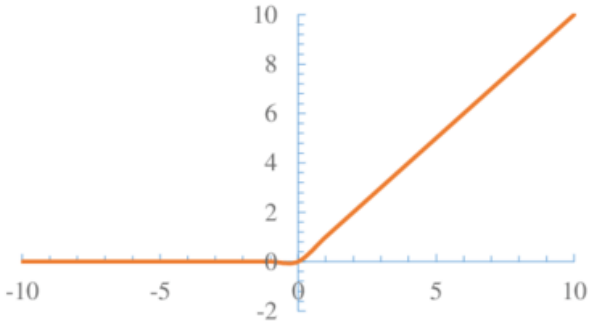


Figura 28. Función ReLU. (Yang et al., 2018)

La unidad ReLU tiene salida 0 para valores menores que 0 y la función identidad para valores mayores que 0. Es la unidad de activación más usada en la práctica (Jarrett, Kavukcuoglu y LeCun, 2009).

3.5.5 Algoritmo de entrenamiento

Un algoritmo de entrenamiento para una red neuronal busca solucionar un problema de optimización. Su objetivo es encontrar los valores adecuados para los pesos (W) y *bias* (b) de la red. Uno de los algoritmos ampliamente utilizados es *stochastic gradient descent* (*SGD*). Este algoritmo aproxima localmente de manera lineal la función a minimizar, y realiza pequeños pasos en la dirección de máximo descenso (Goodfellow et al., 2016).

Si bien *stochastic gradient descent* es un algoritmo que funciona correctamente en general, tiene una serie de problemas que dificultan en ocasiones el entrenamiento de redes neuronales. Uno de los algoritmos que se han creado para resolver estos problemas es *Adam* (Kingma y Ba, 2014). Este algoritmo combina técnicas tomadas de otros métodos como *RMSProp* (Hinton G. , 2012) y *SGD con momentum* para mejorar la velocidad con la que converge en la búsqueda de una solución óptima. *Adam* también reduce la probabilidad de que el algoritmo se detenga en una posición intermedia y no sea capaz de avanzar en la búsqueda de un mínimo local o global.

3.5.6 Regularización

Las redes neuronales son modelos con un gran poder de representación. El gran número de parámetros y capas que se añaden a una red neuronal hace que sea fácil que esta aprenda demasiado bien los datos con los que se está entrenando. En ocasiones, es posible que la red sea capaz de memorizar perfectamente un conjunto de datos de entrenamiento, llegando a clasificarlos a la perfección (100% de exactitud). Sin embargo, al hacer esto, el modelo pierde capacidad de generalización y, al ser evaluado sobre un conjunto de datos no vistos durante el entrenamiento o test set, se puede observar una baja capacidad de predicción (Goodfellow et al., 2016). Este fenómeno se conoce como *sobreajuste* y es un fenómeno común en aprendizaje automático: el algoritmo está modelando demasiado bien los datos de entrenamiento mientras que pierde capacidad de generalización en datos no vistos. Más que aprender las características generales de un conjunto de datos, el algoritmo está fijándose y aprendiendo los detalles particulares de los datos con los que es entrenado.

Las técnicas de regularización intentan solucionar este problema. *Dropout* (Srivastava, 2013) es una técnica bastante moderna y que ha encontrado una gran acogida. La idea consiste en aplicar una salida 0 a un porcentaje de neuronas de la red durante el entrenamiento de manera aleatoria. De este modo, en cada iteración, un número aleatorio de neuronas se desactivará. La probabilidad p de que una neurona se desactive es otro hiperparámetro de la red y es común utilizar valores de 0.25 o 0.5. En este último caso,

significa que durante el entrenamiento la mitad de las neuronas se van desactivando en cada iteración. Dropout impide la memorización de las variables. Al forzar que ciertas neuronas tengan salida 0, la red tiene que aprender nuevas formas de correlacionar las variables. Sin *dropout*, es posible que la red memorice la presencia continua de ciertas neuronas activadas para cierto valor de salida. Con *dropout*, la red es forzada a aprender nuevas relaciones entre neuronas, ya que muchas de las neuronas han quedado apagadas. Esto tiene un efecto regularizador, ya que se le impide a la red memorizar resultados.

3.6 Comparación con otros modelos de aprendizaje automático

Los resultados obtenidos con el modelo de red neuronal son satisfactorios, sin embargo, es importante validarlos frente al desempeño de otros modelos de aprendizaje automático. Esta evaluación comparativa permite conocer si los resultados de las métricas aplicadas al modelo son similares a los que se obtienen con otros algoritmos, o si existe algún modelo que ofrezca una mejor solución. Los algoritmos de *máquinas de vector de soporte* (SVM) y *random forest* son conocidos por su buen desempeño en tareas de clasificación. Por esta razón se entrenará el conjunto de datos con estos dos modelos con el fin de evaluar su rendimiento frente al modelo de redes neuronales.

3.6.1 Máquina de vectores de soporte (SVM)

Las *máquinas de vector de soporte* (SVM) son el primer modelo de aprendizaje automático en emplear un enfoque diferente al probabilístico (Vapnik, 1982). Se basan en un modelado geométrico del problema que se resuelve mediante una optimización con restricciones. El objetivo de las máquinas de vector de soporte es generar varios planos espaciales que separen los ejemplos de acuerdo a sus correspondientes clases. De todas las soluciones propuestas se elige aquella que ofrezca una mejor separación de las clases. En la *Figura 29* se presentan varios planos que ofrecen diferentes soluciones para separar los dos conjuntos de ejemplos representados en color azul y rojo. A la derecha se muestra la solución óptima que ofrece la mejor separación de clases.

Esta es una tarea sencilla cuando se trata de un espacio bidimensional, sin embargo, la complejidad aumenta a medida que se incrementa el número de dimensiones del problema. En el caso de un problema de clasificación binaria, de todos los planos posibles es necesario buscar aquel que proporcione la mayor diferencia entre las dos clases, lo cual se

traduce en la mayor distancia entre los puntos que pertenecen a una clase y a otra. Este plano será el que tendrá una mayor distancia en el conjunto de test y en las predicciones futuras (Cortes y Vapnik, 1995).

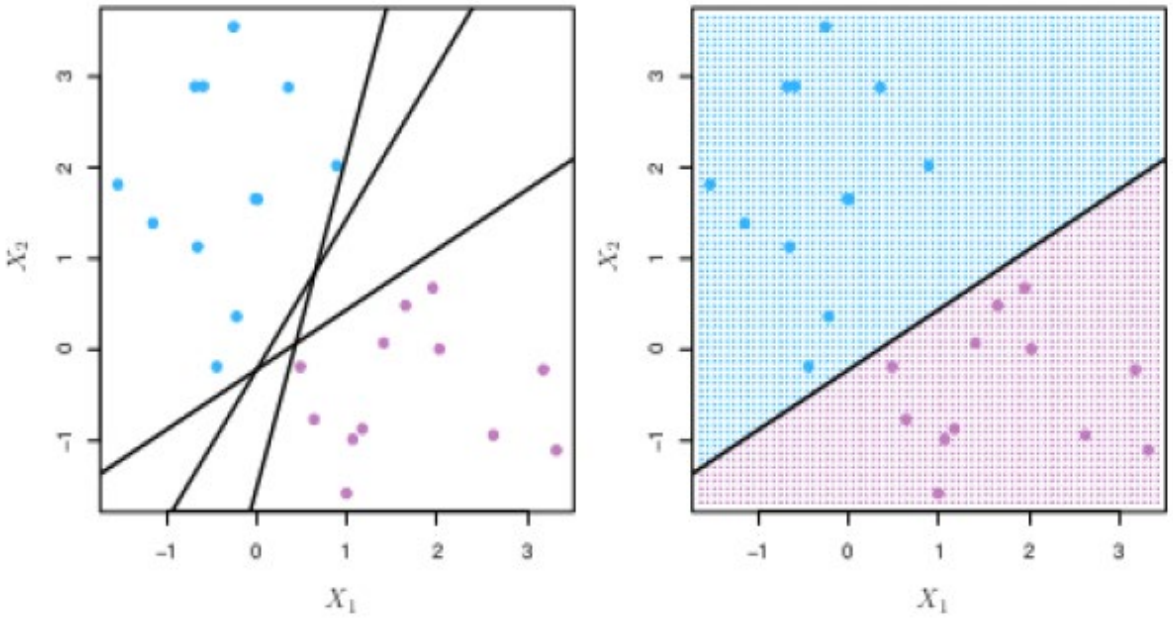


Figura 29. Ejemplo de separación de clases con SVM. (James et al., 2013).

3.6.2 Random Forest

El siguiente modelo se entrenó utilizando el algoritmo de *Random Forest* (Breiman, 2001). Este algoritmo surge como respuesta a uno de los inconvenientes principales de los árboles de decisión, que es su baja capacidad predictiva. La solución para este problema es el uso de un *ensemble* o combinación de varios árboles de decisión. Los ensembles de árboles se pueden crear empleando técnicas de *bagging* o *boosting*. Los modelos de random forests se basan en la combinación de árboles haciendo uso de métodos de bagging (Breiman, 1994). Los métodos de *bagging* permiten entrenar varios árboles por separado, llevar a cabo las predicciones sobre los datos de test y luego obtener la predicción final midiendo la moda de todas las predicciones. Es decir, la predicción global es la clase que más veces ocurre a lo largo de todas las predicciones. Para el caso de los problemas de regresión la predicción es la media de las predicciones de cada uno de los árboles entrenados.

El modelo de *random forest* hace una combinación del método de *bagging,* con una técnica de selección de variables aleatorias con el fin de añadir diversidad a los árboles de decisión. De esta manera todos los arboles son entrenados con un conjunto diferentes de variables y cada variable tiene la oportunidad de aparecer en más de un modelo. Se trata de un

algoritmo que combina versatilidad y potencia en un solo enfoque. A la hora de construir cada uno de los árboles se utiliza una porción pequeña y aleatoria de las variables de entrada disponibles, por lo general, este número se define como $\sqrt{p}$ siendo p el número de variables de entrada. Otra característica del algoritmo de random forest es el uso de técnicas de *bootstrapping*, que permiten realizar re muestreos en el conjunto de datos. De esta manera se logra ampliar el número de datos empleados en el entrenamiento reutilizando los ejemplos de forma aleatoria. Este modelo se basa en la utilización de un gran número de árboles.

Capítulo 4 – Experimentos y análisis de resultados

En este capítulo se describen los experimentos llevados a cabo para entrenar, optimizar y evaluar el modelo con redes neuronales para el pronóstico de la enfermedad renal crónica. También se realiza la comparación con otros modelos de aprendizaje automático, como lo son SVM y random forest. Finalmente se realiza la implementación del modelo entrenado sobre el conjunto total de datos de la población colombiana y se analizan los resultados obtenidos.

4.1 Modelo inicial de red neuronal

El modelo inicial de la red neuronal se implementó haciendo uso de la librería *keras* en *Python*. Esta librería facilita la creación y evaluación de clasificadores con redes neuronales. La clase *Sequential* permite agregar nuevas capas a la red, indicando para cada una el número de neuronas, así como la función de activación a implementar. La *Figura 30* presenta el código desarrollado para la creación de la red neuronal. Se puede observar que se agregan 4 capas, correspondientes a las 3 capas ocultas y a la capa de salida. No se agrega la capa de entrada, pero se indica en la primera capa oculta que existe una entrada con una dimensión de 7.492 elementos. Para todas las capas se aplicará por defecto una función de activación de tipo sigmoide, en la que la salida de cada nodo únicamente puede tomar un valor entre 0 y 1.

```
#Definición del modelo
model = Sequential()
model.add(Dense(units=500, activation='sigmoid', input_dim=7492))
model.add(Dense(units=100, activation='sigmoid'))
model.add(Dense(units=50, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))

#compilación del modelo
model.compile(loss='binary_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Figura 30. Implementación en Keras del modelo inicial de la red.
Fuente: Elaboración propia en Python 3.7.0.

Una vez definido el modelo se debe compilar haciendo uso de un algoritmo de optimización. Es común el uso del algoritmo de *gradient descent*, aunque en la práctica se utiliza una variación de este algoritmo conocido como *stochastic gradient descent (SGD)* que es menos costoso computacionalmente (Goodfellow et al., 2016). La función *compile* de *keras* recibe como parámetros el optimizador del modelo, en este caso *SGD*, la función de pérdida y la métrica empleada para la evaluación del modelo (Gulli et al., 2017). Como función de pérdida se empleará *binary crossentropy* y como métrica de desempeño del modelo se utilizará la exactitud, o *accuracy* en inglés. La *Figura 31* presenta el resumen de la red neuronal desarrollada en *keras*, junto con el número de parámetros a entrenar por cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_44 (Dense)             (None, 500)               3746500
_________________________________________________________________
dense_45 (Dense)             (None, 100)               50100
_________________________________________________________________
dense_46 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 51
=================================================================
Total params: 3,801,701
Trainable params: 3,801,701
Non-trainable params: 0
_________________________________________________________________
```

Figura 31. Resumen del diseño de la red en Keras.
Fuente: Elaboración propia en Python 3.7.0.

El número de parámetros en cada capa corresponde al número de nodos de la capa anterior, multiplicado por el número de nodos de la capa actual, más un valor de *bias* por cada nodo de la capa actual, por ejemplo, para la primera capa oculta el número de parámetros esta dado por el número de nodos de la capa de entrada (7.492) multiplicado por el número de nodos de la primera capa oculta (500), más 500 valores de *bias* correspondientes a cada nodo de la capa actual. Esto suma un total de 3.746.500 parámetros para la primera capa oculta. Para las restantes capas el número de parámetros es de 50.100, 5.050 y 51, arrojando un valor total de 3.801.701 parámetros para toda la red.

El siguiente paso corresponde al entrenamiento de la red, que se implementa con la función *fit* de *keras*. Esta función toma como parámetros los conjuntos de datos que contienen las variables *x* y *y* del conjunto de entrenamiento, el porcentaje de los datos que se utilizarán como conjunto de validación, el número de épocas y el tamaño de lote o *batch*. El tamaño del conjunto de validación corresponde al 20% de los datos, el número de épocas en las que

se realizará el entrenamiento corresponde a 10 iteraciones y el tamaño de lote, es decir el número de ejemplos en los que se subdividirá el conjunto de datos para su entrenamiento, es de 1.000. La *Figura 32* muestra el uso de la función fit para el entrenamiento de la red. La salida de la función corresponde a la información de las métricas de desempeño para cada época o iteración del algoritmo. Estos datos se emplearán para comparar el desempeño de cada modelo e identificar cual presenta mejoras con respecto a los demás o cuales tienen problemas en el entrenamiento, como puede ser *sobreajuste*.

```
#entrenamiento del modelo
history_01=model.fit(entrenamientoX, entrenamientoY,
                     validation_split=0.2, epochs=10, batch_size=1000)
```

Figura 32. Entrenamiento del modelo en keras.
Fuente: Elaboración propia en Python 3.7.0.

Una vez se ha entrenado el modelo se deben utilizar los datos de prueba para identificar el desempeño obtenido con el modelo actual. En este caso se hace uso de la función *evaluate* de keras, a la que se le envían como parámetros los dataframes que contienen las variables *x* y *y* del conjunto de datos de prueba. Como resultado de la función se obtiene el valor de exactitud para el modelo actual, que en este caso corresponde a un 49.91%. Este es un resultado muy bajo que debe ser corregido haciendo ajustes en los parámetros empleados en el entrenamiento del modelo. En los siguientes apartados se realizará una optimización del modelo inicial donde se modifican algunos parámetros como la función de activación, el algoritmo de entrenamiento y se agregan otras características como normalización y *early stopping*.

4.2 Modelo con función de activación ReLU

Durante el entrenamiento del modelo inicial de la red se obtuvo un valor de exactitud demasiado bajo, por lo tanto, se optó por cambiar la función de activación *sigmoide* por una función *ReLU*. El valor de exactitud obtenido con este ajuste en el modelo fue del 59.96%. En las siguientes figuras se compara el valor del error de la red (loss), así como de la exactitud (accuracy) para cada modelo. También se muestran las diferencias entre los conjuntos de entrenamiento y validación.

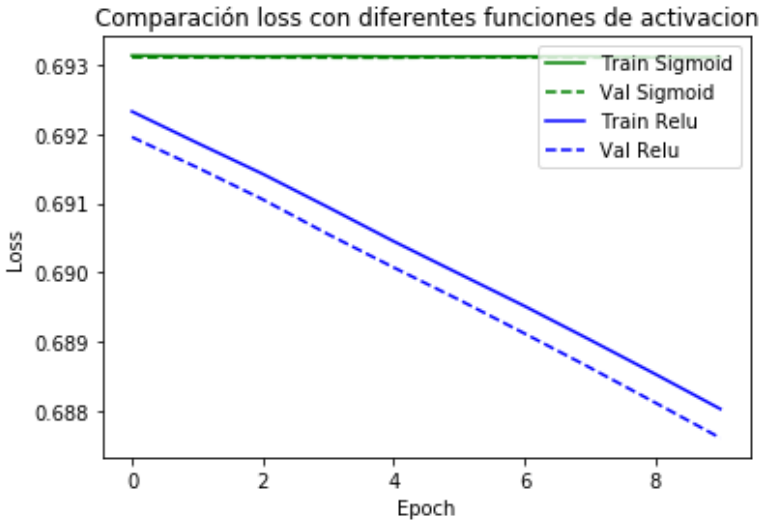


Figura 33. Comparación del error obtenido con diferentes funciones de activación
Fuente: Elaboración propia en Python 3.7.0.

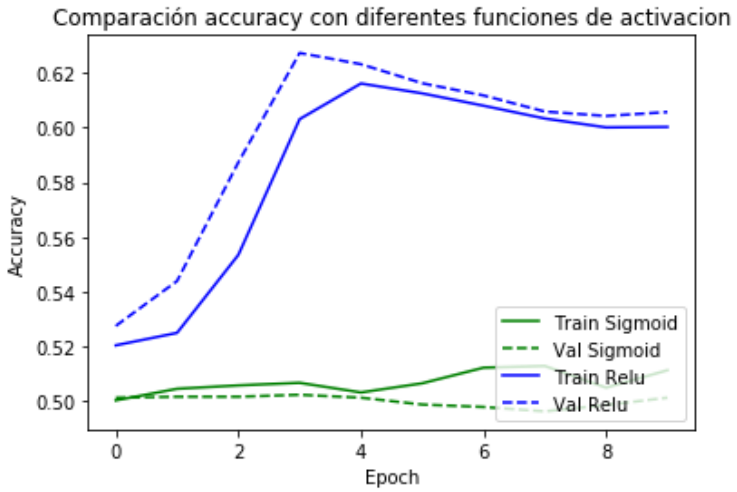


Figura 34. Comparación de la *exactitud* obtenida con diferentes funciones de activación.
Fuente: Elaboración propia en Python 3.7.0.

Como se puede observar en las gráficas anteriores, la función de activación ReLU permite a la red disminuir el valor del error y a la vez mejorar considerablemente la medida de exactitud, aunque este valor aún se encuentra por debajo del umbral deseado. Por esta razón en el siguiente apartado se intentará mejorar la métrica de desempeño realizando cambios en el algoritmo de entrenamiento.

4.3 Modelo con algoritmo de optimización Adam

Con el fin de mejorar el desempeño de la red se cambió el algoritmo de optimización de *SGD* a *Adam*. Este cambio permitió obtener un valor de exactitud de 92.65%. Este valor mejora considerablemente la métrica obtenida anteriormente con *SGD* igual a 59.96%. También supera el umbral del 90% propuesto en los objetivos del proyecto.

En la siguiente figura se puede observar la disminución en la medida del error tanto en el conjunto de entrenamiento, como en el de validación utilizando optimización con Adam, frente a la obtenida anteriormente con *SGD*.

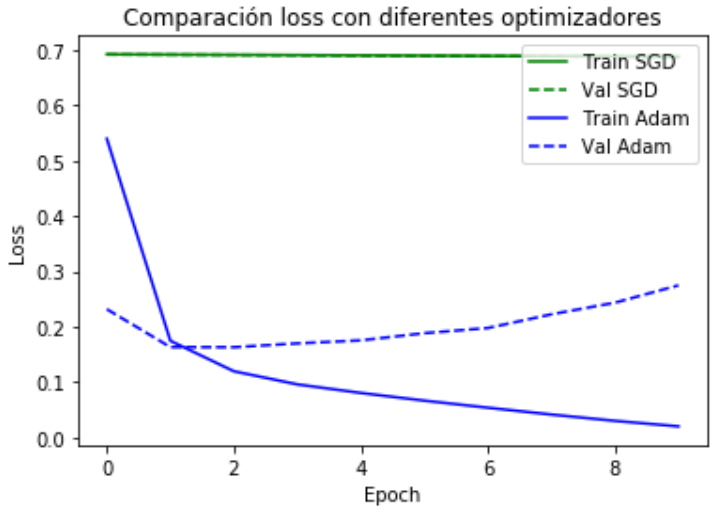


Figura 35. Comparación del error obtenido con diferentes optimizadores.
Fuente: Elaboración propia en Python 3.7.0.

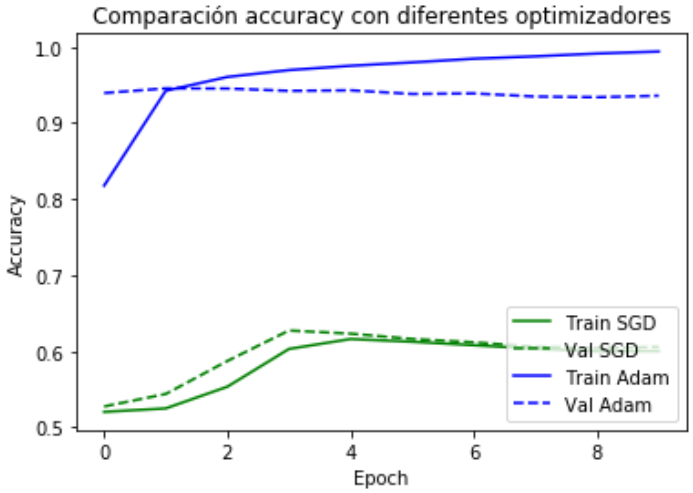


Figura 36. Comparación de la exactitud obtenida con diferentes optimizadores.
Fuente: Elaboración propia en Python 3.7.0.

La *Figura 35* muestra como la curva correspondiente al error en el conjunto de entrenamiento disminuye hasta valores cercanos a 0, mientras que la curva correspondiente al error obtenido con el conjunto de datos de validación aumenta. La *Figura 36* muestra un aumento en el valor de exactitud que llega a valores por encima del 90%. Sin embargo, se puede observar que en el modelo tiende a presentar sobre ajuste, también conocido como *sobreajuste*. Esto se debe a que el algoritmo está memorizando los datos de entrenamiento y no es capaz de generalizar lo aprendido al conjunto de validación.

4.4 Modelo con regularización

Una vez aplicada la técnica de dropout a la salida de cada capa de la red, con un valor de hiperparámetro igual a 0.5, se obtiene un valor de exactitud igual a 93.71%, mejorando ligeramente frente al obtenido anteriormente sin aplicar regularización. Las siguientes figuras muestran la comparación entre los valores de error y exactitud para los modelos con regularización y sin regularización. En la *Figura 37*, se observa que el modelo entrenado aplicando dropout (en color azul) disminuye la presencia de sobreajuste, aunque no lo puede evitar del todo.

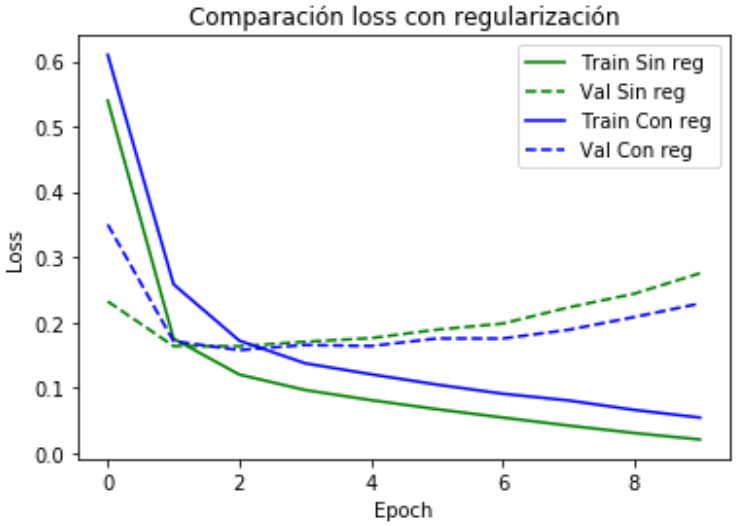


Figura 37. Comparación del error aplicando regularización.
Fuente: Elaboración propia en Python 3.7.0.

La *Figura 38* también demuestra la disminución del sobreajuste y un ligero aumento en el valor de exactitud en el conjunto de validación del modelo entrenado empleando *dropout*.

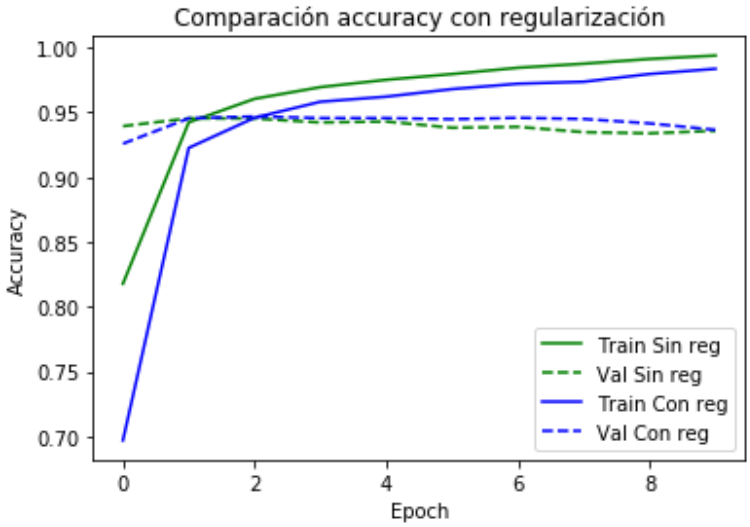


Figura 38. Comparación de la exactitud aplicando regularización.
Fuente: Elaboración propia en Python 3.7.0.

4.5 Modelo final de red neuronal

Otra técnica muy utilizada en la práctica para prevenir sobre ajuste es la de *early stopping* (Bishop, 1995). Esta técnica permite identificar la iteración o época en que la red comienza a caer en *sobreajuste* y detiene el entrenamiento en ese momento. La librería *keras* permite agregar esta técnica al modelo de redes neuronales adicionando un parámetro de *callback* en la función *fit* vista anteriormente (Gulli et al., 2017). En la *Figura 39* se puede observar el código en Python correspondiente al modelo final de la red neuronal entrenada.

En este código se pueden observar los diferentes ajustes realizados al modelo. Entre estos ajustes se encuentra el cambio de la función de activación *sigmoide* por la función *ReLU.* Se debe tener en cuenta que en la capa de salida se permite el uso de la función sigmoide, tenido en cuenta que se quiere obtener un valor de probabilidad con valores entre 0 y 1. Otros ajustes realizados son el cambio de optimizador de *sgd* a *adam*, la inclusión de la técnica de regularización *dropout* y la creación de una función de *early stopping*.

```
#Definición del modelo
model_final = Sequential()
model_final.add(Dense(units=500, activation='relu', input_dim=7492))
model_final.add(Dropout(0.5))
model_final.add(Dense(units=100, activation='relu'))
model_final.add(Dropout(0.5))
model_final.add(Dense(units=50, activation='relu'))
model_final.add(Dropout(0.5))
model_final.add(Dense(units=1, activation='sigmoid'))

#compilación del modelo
model_final.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

#Agrega early stopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

#entrenamiento del modelo
history_final=model_final.fit(entrenamientoX, entrenamientoY,
                              validation_split=0.2, epochs=3, batch_size=1000, callbacks=[es])
```

Figura 39. Modelo final de la red neuronal en keras.
Fuente: Elaboración propia en Python.

La exactitud obtenida mediante este modelo corresponde a un valor de 95%. La *Figura 40* presenta la evolución del error en el modelo final de la red neuronal. Se puede observar que el número de épocas de entrenamiento disminuyó a unas pocas y que el error del conjunto de entrenamiento y del conjunto de pruebas converge en un punto de la gráfica.

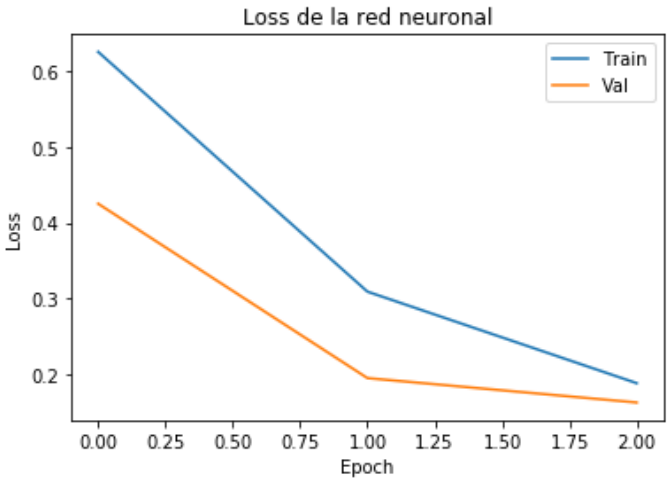


Figura 40. Error en el modelo final de red neuronal.
Fuente: Elaboración propia en Python 3.7.0.

De manera similar la *Figura 41* presenta la exactitud del modelo final, donde se observa que la métrica converge para los dos conjuntos de datos y alcanza un valor de 95% muy cercano al obtenido con los datos de prueba.

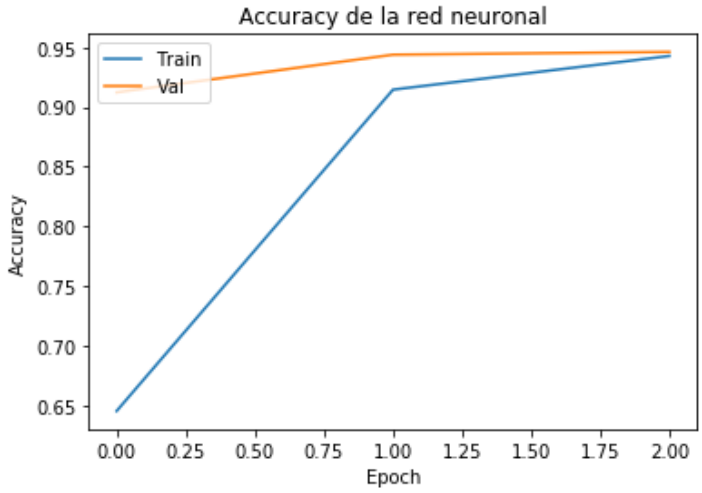


Figura 41. Exactitud en el modelo final de red neuronal.
Fuente: Elaboración propia en Python 3.7.0.

4.6 Evaluación del modelo de red neuronal

El objetivo de las métricas para la evaluación de algoritmos de clasificación es identificar la capacidad de predicción del modelo. Para lograr este objetivo se comparan las clases predichas por el algoritmo para cada ejemplo del conjunto de test con el valor real de la clase y de esta manera se identifica si el modelo logró clasificar correctamente el dato. Una técnica ampliamente usada es la matriz de dispersión, en la que se realiza el conteo de los ejemplos clasificados correcta e incorrectamente, agrupándolos en verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos (Fawcett, 2006).

La *Figura 42* muestra de forma gráfica la matriz de confusión para el modelo final de la red neuronal. El eje x corresponde los valores predichos por el modelo, y el eje y a los valores reales. La posición (1,1) corresponde a al número de ejemplos verdaderos positivos, la posición (0,0) a los verdaderos positivos, la posición (0,1) los falsos negativos y la posición (1,0) los falsos positivos. Como se puede observar en el gráfico, la red clasificó correctamente el 95% de los casos y solamente falló en el 5%. De este porcentaje de errores la mayor parte corresponde a falsos negativos, es decir que el modelo está pronosticando que la persona está sana cuando en realidad si presenta la enfermedad.

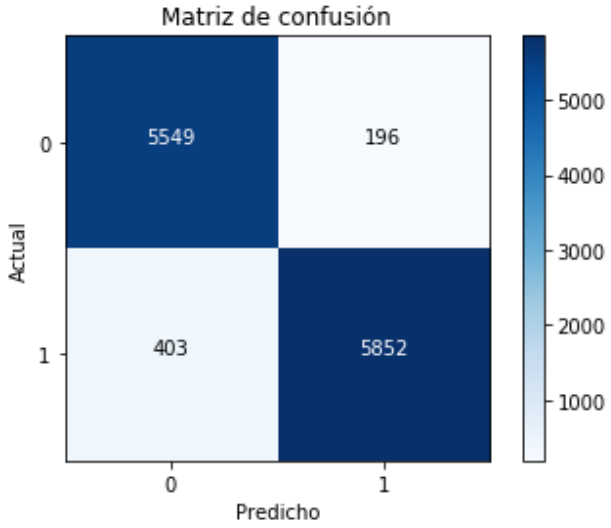


Figura 42. Matriz de confusión.
Fuente: Elaboración propia en Python 3.7.0.

A partir de la matriz de dispersión se pueden calcular métricas de un solo valor como lo son:

Exactitud o accuracy: Representa la proporción del número de predicciones correctas entre el número total de predicciones.

Sensibilidad y Especificidad: La sensibilidad de un modelo es la proporción de ejemplos positivos correctamente clasificados. La especificidad de un modelo indica la proporción de los ejemplos negativos correctamente clasificados.

Precision y recall: La precisión indica la proporción de ejemplos que son verdaderamente positivos, mientras que la métrica de recall indica que tan completos son los resultados.

F-measure: también conocida como F1, es una métrica que combina precisión y recall utilizando la media armónica.

La *Tabla 2* resume el valor de las métricas obtenidas con este modelo:

Tabla 2. Métricas de evaluación del modelo

Métrica	**Valor**
Sensibilidad	97%
Especificidad	93%
Precisión	94%
Exhaustividad	97%
Valor-F	95%
Exactitud	95%
Área bajo la curva (AUC)	98%

Fuente: Elaboración propia

En estas métricas se confirma, por una parte, la capacidad de predicción del clasificador obtenida mediante la métrica de exactitud, y por otra parte una tendencia del modelo para predecir con mayor precisión los ejemplos positivos respecto a los negativos. Esto se puede observar teniendo en cuenta que el valor de sensibilidad, que mide la proporción de ejemplos positivos correctamente clasificados, es más alto que el valor de especificidad, que mide la proporción de ejemplos negativos correctamente clasificados.

En el caso de los algoritmos de clasificación binaria, se puede utilizar una métrica conocida como área bajo la curva o en inglés *area under the curve* (AUC). Esta métrica se utiliza para determinar el balance entre la detección de verdaderos positivos y evitar los falsos positivos. Para ello, se muestra la proporción de detección de los verdaderos positivos en el eje y, y la proporción de los falsos positivos en el eje x. A continuación, se muestra la curva ROC obtenida junto con su valor de AUC.

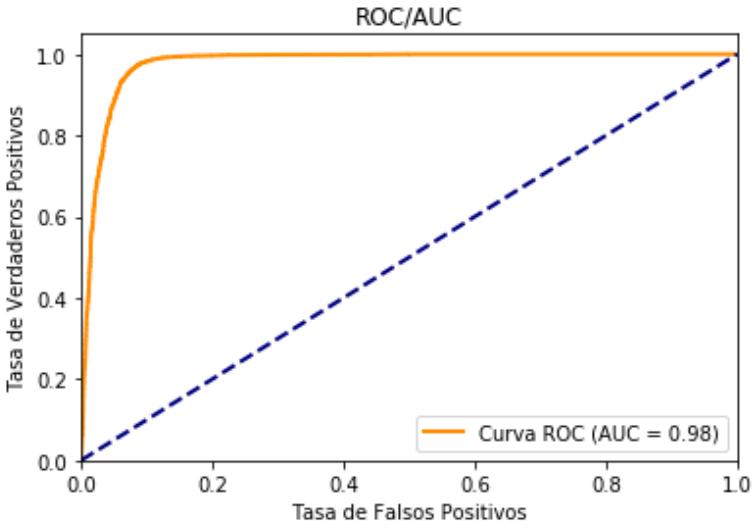


Figura 43. Curva ROC/AUC.
Fuente: Elaboración propia en Python 3.7.0.

4.7 Modelo con SVM

La implementación del algoritmo de SVM se realizó en Python haciendo uso de la función *SVC* incluida en la librería *scikit learn* versión 0.21. Esta librería ofrece una gran cantidad de funcionalidades para el entrenamiento de modelos de aprendizaje automático. La función *SVC* permite realizar el entrenamiento de un clasificador ya sea binario o multiclase, y recibe como entrada los hiperparámetros propios del modelo SVM. Uno de los hiperparámetros más importantes es la función de coste (C) que determina el tamaño de la margen del plano. Un valor de C más grande hace el margen más pequeño y un valor C más pequeño hace el margen más grande. Otro hiperparámetro que se puede especificar es el tipo de *kernel* a implementar. El uso de *kernels* permite obtener fronteras de decisión no lineales por medio de transformaciones matemáticas sin necesidad de tener que realizar transformaciones con polinomios (James et al., 2013). La *Figura 44* muestra un resumen de los parámetros con los que se realizó el entrenamiento del modelo con SVM.

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Figura 44. Parámetros de entrenamiento del modelo de SVM.
Fuente: Elaboración propia en Python 3.7.0.

Una vez aplicado el algoritmo de SVM, se utilizó el conjunto de datos de test para realizar la evaluación del modelo. El valor de exactitud resultante es del 61%, un valor bastante inferior al obtenido con la red neuronal. La matriz de confusión de la *Figura 45* demuestra que un gran número de ejemplos se clasificaron de forma incorrecta. Hay un número elevado de falsos negativos, es decir ejemplos que corresponden a personas con ERC que fueron clasificadas como NO ERC.

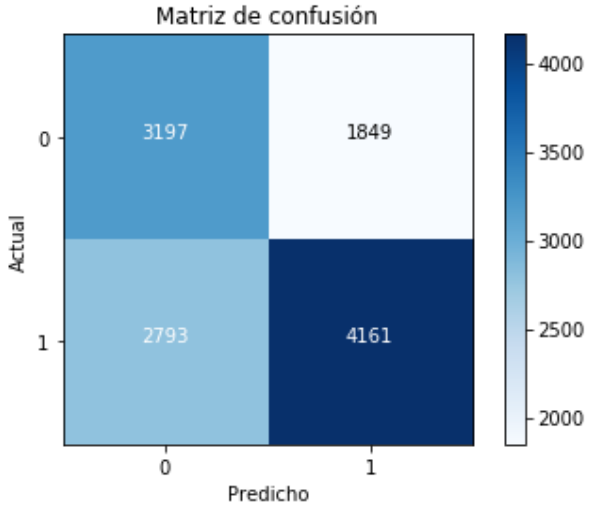


Figura 45. Matriz de confusión para el modelo entrenado con SVM.
Fuente: Elaboración propia en Python 3.7.0.

El modelo se está comportando de forma poco arriesgada al clasificar como negativos ejemplos que pertenecen a la clase positiva. Este comportamiento afecta a la curva ROC que se presenta en la *Figura 46*, donde se indica una medida de *área bajo la curva* (AUC) igual a 0.61. La AUC confirma los resultados de la matriz de confusión que demuestran que el clasificador no está realizando su tarea de forma adecuada.

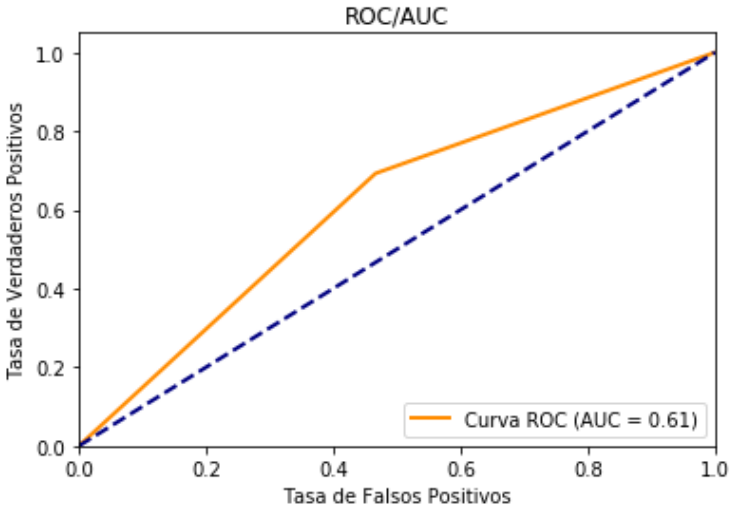


Figura 46. Curva ROC para el modelo entrenado con SVM.
Fuente: Elaboración propia en Python 3.7.0.

4.8 Modelo con random forest

La implementación del algoritmo de random forest se llevó a cabo a través de la función *RandomForestClassifier* incluida en la librería *scikit learn* versión 0.21. Esta función permite entrenar clasificadores de una manera muy eficiente, indicando el número de árboles que se desean crear, así como otros hiperparámetos importantes para optimizar el desempeño final del modelo. La *Figura 47* muestra un resumen de los parámetros con los que se realizó el entrenamiento del modelo con SVM.

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Figura 47. Parámetros de entrenamiento del modelo de Random Forest.
Fuente: Elaboración propia en Python 3.7.0.

Una característica muy importante de Random Forest, es que, si bien se trata de un modelo de caja negra, es posible conocer la importancia que el algoritmo le otorga a cada variable de entrada. La importancia mide el impacto que tiene cada variable en la predicción final del modelo. Para el caso del modelo entrenado con los datos demográficos y de atenciones médicas, se obtuvo la distribución de probabilidades indicada en la *Figura 48*. En esta imagen se encuentran las 20 variables con mayor importancia para el modelo. En el primer lugar se encuentra la variable Edad, seguida por un conjunto de diagnósticos entre los que se encuentran E119, que corresponde a diabetes mellitus, M255, correspondiente a dolor en articulaciones, N390, infección de vías urinarias y I10X, hipertensión arterial. Esto coincide con la teoría médica que señala estas variables como los principales factores de riesgo de la enfermedad renal crónica.

variable	importancia
Edad	0.055603
E119	0.035742
M255	0.033023
N390	0.030263
I10X	0.028844
J449	0.021672
M542	0.020566
E109	0.019875
R104	0.018675
R51X	0.016306
I839	0.010781
R520	0.010668
H400	0.010570
M791	0.010283
M545	0.010094
M544	0.009683
K30X	0.009454
K053	0.008851
K295	0.008258
H811	0.007945

Figura 48. Importancia de las variables en el modelo entrenado con Random Forest.
Fuente: Elaboración propia en Python 3.7.0.

Una vez aplicado el algoritmo de Random Forest, se utilizó el conjunto de datos de test para realizar la evaluación del modelo. El valor de exactitud resultante es del 92%, un valor muy cercano al obtenido con la red neuronal, que fue del 95%. La matriz de confusión de la *Figura 49* demuestra que la mayor parte de los ejemplos se clasificaron de forma adecuada.

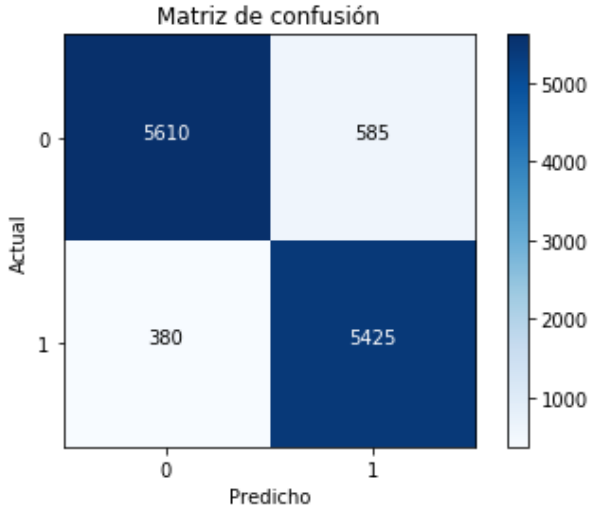


Figura 49. Matriz de confusión para el modelo entrenado con random forest.
Fuente: Elaboración propia en Python 3.7.0.

A diferencia de la red neuronal, el algoritmo de Random Forest actuó de forma un poco más arriesgada al predecir un mayor número de ejemplos como positivos cuando en realidad eran negativos, es decir que los clasifico como ERC cuando el paciente no presentaba la enfermedad. Al tener una tasa de falsos positivos más alta que el modelo de red neuronal, su valor de AUC es menor, obteniendo un valor del 92% como se indica en la *Figura 50*.

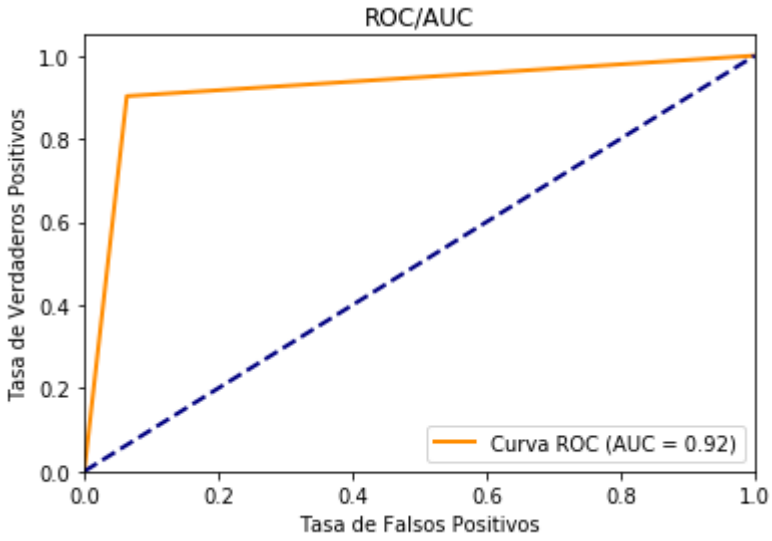


Figura 50. Curva ROC para el modelo entrenado con Random Forest.
Fuente: Elaboración propia en Python 3.7.0.

4.9 Comparación de los modelos

La *Tabla 3* presenta una comparación de las principales métricas obtenidas luego de aplicar los modelos de redes neuronales, SVM y Random Forest. La tabla muestra que el mejor clasificador es la red neuronal, teniendo en cuenta un valor de exactitud del 95%. Esta es seguida por Random Forest con una exactitud del 92%, y en último lugar se encuentra SVM con un valor del 61%.

Tabla 3. Comparación de métricas con otros modelos

Métrica	**Red Neuronal**	**SVM**	**Random Forest**
Sensibilidad	97%	69%	90%
Especificidad	93%	53%	94%
Precisión	94%	60%	93%
Exhaustividad	97%	69%	90%
Valor-F	95%	64%	91%
Exactitud	**95%**	**61%**	**92%**
Área bajo la curva (AUC)	98%	61%	92%

Fuente: Elaboración propia

La métrica de *sensibilidad*, que mide la proporción de ejemplos positivos clasificados correctamente muestra un mejor desempeño de la red neuronal frente a random forest, y de estos dos frente al SVM. En cuanto a la *especificidad*, el modelo que clasifica mejor los ejemplos negativos es random forest, aunque el valor obtenido por la red neuronal es bastante cercano. Igual sucede con la métrica de *precisión* que indica la proporción de ejemplos que son verdaderamente positivos. Los valores de *recall* favorecen por un amplio margen a la red neuronal, que obtiene un valor de 97, frente a un 90 de random forest y un 69 de SVM. El *recall* mide que tan completos son los resultados y es similar a la sensibilidad del modelo. El valor de *F-measure* corresponde a un balance entre precisión y recall, y simplifica el rendimiento de un algoritmo de clasificación en una única métrica. La red neuronal obtiene un valor de F-measure de 95, random forest de 91 y SVM de 64. Finalmente, el área bajo la curva (AUC) del modelo de redes neuronales supera de nuevo a los otros algoritmos debido a su tendencia a identificar correctamente los verdaderos positivos y evitar los falsos positivos.

El conjunto de métricas aplicadas demuestra que el modelo entrenado con redes neuronales logró un excelente desempeño, superando el umbral propuesto en los objetivos del proyecto. Por otra parte, el modelo entrenado con random forest obtiene también buenos resultados, muy cercanos a los de la red neuronal y se contempla como una buena alternativa a este último modelo. El desempeño del modelo de SVM para este caso se considera muy bajo y se descarta su implementación.

4.10 Implementación del modelo de red neuronal

Una vez realizado el entrenamiento y evaluación del modelo de redes neuronales, se llevó a cabo su aplicación al conjunto completo de datos de la población colombiana, con el fin de pronosticar nuevos casos de enfermedad renal crónica. El conjunto total de personas se determinó teniendo en cuenta aquellas personas que recibieron al menos una atención en salud en el periodo de tiempo comprendido entre los años 2015 y 2018. Esta cifra corresponde a un total de 39.277.086 personas. A continuación, se identificaron los diagnósticos realizados a estas personas durante todo el periodo de tiempo del que se dispone información en la base de datos de RIPS, es decir para los años 2009 a 2018. Junto con el diagnóstico se identificó el número de atenciones médicas prestadas a la persona por cada diagnóstico. Este conjunto de datos se extrajo en un archivo de tipo csv, obteniendo un volumen de 371.851.278 registros con la combinación persona-diagnóstico.

Dada las dificultades presentadas para cargar dicho archivo en Python y realizar la conversión a la estructura requerida por el modelo de datos, fue necesario particionar el archivo en un grupo de 392 archivos con 100.000 personas cada uno. Sin embargo, se determinó que el tiempo requerido para leer cada archivo, transformar su conjunto de datos y aplicar el modelo era de 1.984 segundos en promedio. Para el total de 392 archivos este tiempo correspondía a un total de 216 horas, es decir 9 días aproximadamente.

Con el fin de reducir el tiempo de implementación del modelo se construyó un programa en lenguaje C, que llevara a cabo la tarea de generación de un conjunto de archivos con la estructura de datos requerida por el modelo de redes neuronales. Esta rutina permitía al programa en Python tomar los datos en el formato requerido y aplicar el modelo entrenado, sin tener que llevar a cabo la transformación de los datos. Se determinó que cada archivo debería contener un conjunto de personas similar a las requeridas para entrenar el modelo, es decir 40.000 personas. De esta manera el número de archivos generado por el programa en C fue de 982. La *Figura 51* presenta un diagrama del proceso llevado a cabo para la implementación del modelo.

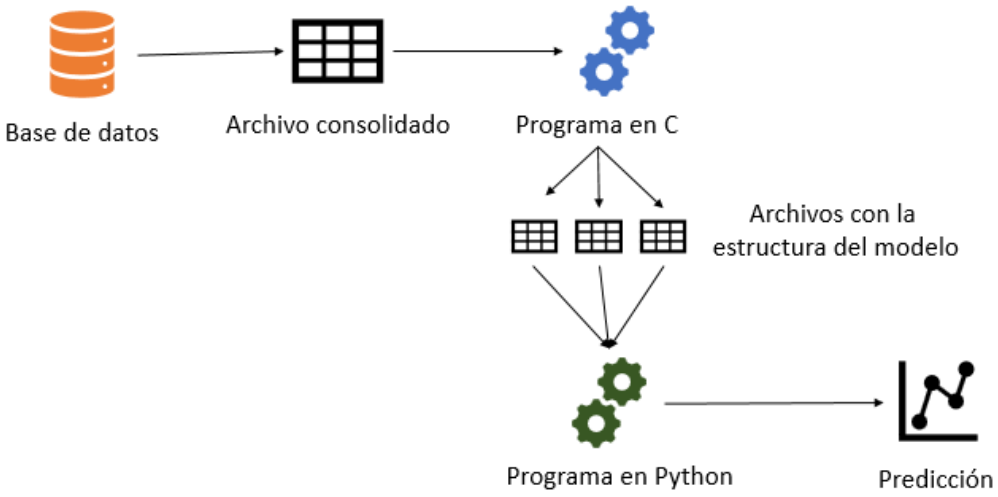


Figura 51. Diagrama de implementación del modelo.
Fuente: Elaboración propia.

El tiempo requerido para la generación de cada archivo y la aplicación del modelo de redes neuronales fue de 90 segundos en promedio. El tiempo total para el procesamiento de los 982 archivos fue de 24.5 horas, logrando una reducción de tiempo del 89% respecto al modelo de implementación planteado inicialmente.

4.11 Análisis de resultados

Una vez realizado el pronóstico para el conjunto de 39.277.086 personas, el modelo identifico que 3.494.516 personas se encuentran en riesgo de desarrollar enfermedad renal crónica en Colombia. Esta cifra corresponde al 7% del total de la población colombiana. La *Figura 52* muestra la distribución por sexo de las personas a quienes se les pronosticó riesgo de desarrollar ERC. En esta grafica se puede identificar que el 68.08% corresponde a mujeres, y el 31.92% restante corresponde a hombres.

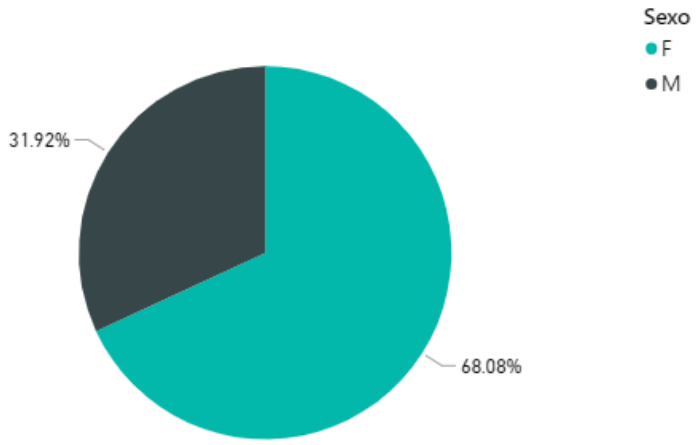


Figura 52. Distribución de personas en riesgo de desarrollar ERC por sexo.
Fuente: Elaboración propia en Power BI.

La *Figura 53*, presenta la distribución de estas personas por edad. Se puede observar pequeños grupos aislados en edades menores a 25 años, un crecimiento suave entre los 25 y los 45 años y un incremento importante a partir de los 45 años, cuyo pico se centra en los 60 años. A partir de esta edad comienza a disminuir el número de personas.

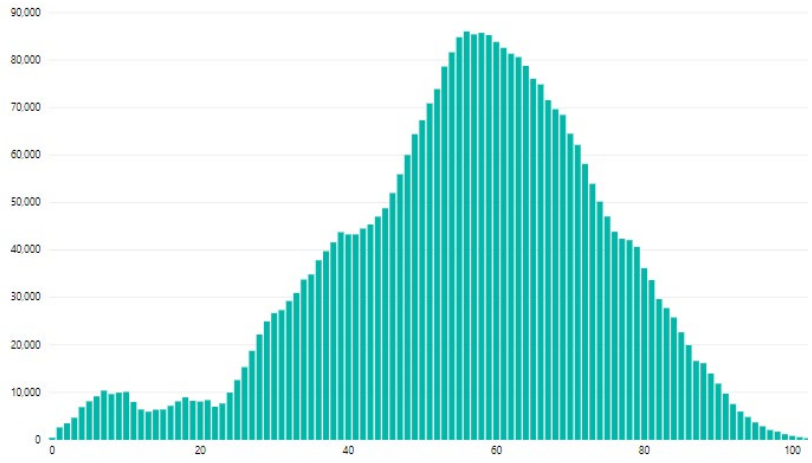


Figura 53. Distribución de personas en riesgo de desarrollar ERC por edad.
Fuente: Elaboración propia en Power BI.

Finalmente, en las *Figura 54 y 55,* se presenta la distribución de personas de acuerdo con su lugar de residencia, en este caso a nivel de departamento. En la primera imagen se presenta el porcentaje de personas con pronóstico de enfermedad renal crónica. El porcentaje se obtiene tomando como numerador el número de personas identificadas en cada departamento, y como denominador la población total del departamento. Los departamentos que presentan un mayor porcentaje de personas con ERC son Bogotá D.C., Antioquia, Risaralda, Caldas, Santander, Atlántico, Boyacá y Quindío. Se puede observar valores más altos en la región central de país, especialmente en los departamentos pertenecientes al eje cafetero.

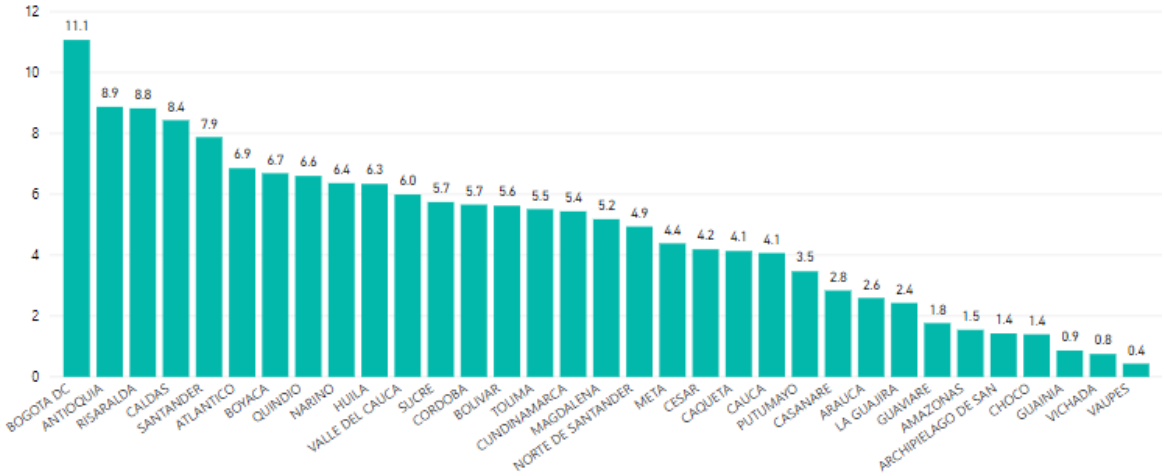


Figura 54. Distribución de personas en riesgo de desarrollar ERC por geografía.
Fuente: Elaboración propia en Power BI.

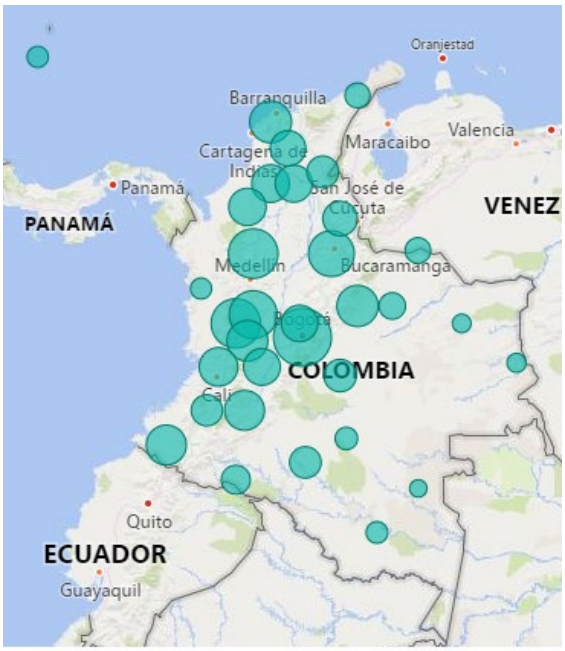


Figura 55. Mapa de cobertura de la población en riesgo de desarrollar ERC.
Fuente: Elaboración propia en Power BI.

Conclusión y trabajo futuro

La ERC es una enfermedad que afecta aproximadamente al 10% de la población mundial, siendo una patología silenciosa que no presenta síntomas hasta que ya se encuentra en un estado avanzado (Organización Panamericana de la Salud, 2015). El tratamiento para la ERC además de ser altamente invasivo e impactar negativamente en la calidad de vida del paciente, también es bastante costoso para los sistemas de salud (Fondo Colombiano de Enfermedades de Alto Costo, 2017). En la actualidad las redes neuronales, y en general los algoritmos de aprendizaje automático han comenzado a utilizarse de forma exitosa como herramientas de apoyo para el diagnóstico y detección temprana de enfermedades. Estos algoritmos se entrenan con grandes volúmenes de datos, que incluyen imágenes diagnósticas, pruebas de laboratorio e historias clínicas. En el presente trabajo se propuso el uso de una red neuronal para el pronóstico de la enfermedad renal crónica en la población colombiana. Para el entrenamiento del clasificador se tomó la información de dos grupos poblacionales, por una parte, pacientes a quienes se les diagnosticó ERC durante el año 2018, y, por otra parte, una muestra de personas sanas. Para ambos grupos de población se recopiló la información demográfica, compuesta por las variables de sexo, edad, lugar de residencia y etnia, junto con el historial de diagnósticos de enfermedades, obtenidos de la base de datos del RIPS (registro individual de prestación de servicios en salud) del ministerio de salud y protección social de Colombia.

Después de entrenar el modelo y aplicar métricas de evaluación de desempeño, se comprobó que la red neuronal entrenada con información demográfica y de diagnósticos de enfermedades, era capaz de pronosticar el riesgo de desarrollar enfermedad renal crónica con una exactitud del 95%, demostrando de esta manera la hipótesis planteada en el presente trabajo. Este resultado se obtuvo entrenando una red neuronal con 5 capas: una capa de entrada con 7.492 neuronas, correspondientes a las variables o características del modelo, 3 capas ocultas con 500, 100 y 50 neuronas respectivamente, y una capa de salida con una única neurona que representa la clase del problema de clasificación binario. Para el entrenamiento de la red se empleó una función de activación *ReLU* en las capas ocultas y función sigmoide para hallar la probabilidad en la capa de salida. Como algoritmo de entrenamiento del modelo se utilizó *Adam*, que demostró una mayor velocidad de convergencia frente a otros algoritmos más tradicionales como *gradient descent* y *stochastic gradient descent* (SGD). Para combatir el efecto del sobreajuste en la red se utilizó regularización haciendo uso de la técnica de *dropout,* en conjunto con *early stopping*.

La evaluación del clasificador se llevó a cabo empleando métricas derivadas de la matriz de confusión, como *sensibilidad*, *especificidad*, *precisión*, *exhaustividad* y área bajo la curva (AUC). El resultado de *sensibilidad*, igual a 97%, y de *especificidad*, igual a 93%, muestran una tendencia de la red para clasificar mejor los ejemplos positivos, frente a los negativos. Esto teniendo en cuenta que la sensibilidad mide la proporción de ejemplos positivos correctamente clasificados, mientras que la especificidad mide la proporción de ejemplos negativos correctamente clasificados. Por otra parte, el valor de AUC igual a 98% demuestra un equilibrio en el comportamiento de la red neuronal, que se esfuerza en predecir la mayor cantidad de casos verdaderos positivos, evitando caer en falsos positivos.

Además de la red neuronal, se realizó el entrenamiento de otros modelos de aprendizaje automático como random forest y máquinas de soporte de vectores (SVM) y se obtuvieron valores de exactitud del 92% y del 61% respectivamente. El modelo de random forest obtiene buenos resultados, muy cercanos a los de la red neuronal y se contempla como una alternativa a este último modelo. A diferencia de la red neuronal, el algoritmo de random forest actuó de forma un poco más arriesgada al predecir un mayor número de ejemplos como positivos cuando en realidad eran negativos, es decir que los clasifico como ERC cuando el paciente no presentaba la enfermedad. Al tener una tasa de falsos positivos más alta que el modelo de red neuronal, la métrica de AUC es menor, obteniendo un valor del 92%. El desempeño del modelo de SVM para este caso se considera muy bajo y se descarta su implementación para pronosticar nuevos casos de ERC.

El clasificador con redes neuronales demostró su capacidad para aprender a identificar factores de riesgo de la enfermedad y luego aplicar este conocimiento en el pronóstico de nuevos casos. El número de personas a quienes el modelo predice en riesgo de desarrollar enfermedad renal crónica en Colombia es de 3.494.516, es decir un 7% del total de la población. El 68.08% de estas personas corresponde a mujeres, y el 31.92% restante corresponde a hombres. La distribución por edad muestra una mayor concentración de ERC en personas entre 40 y 80 años, teniendo el pico más alto hacia los 60 años. Los departamentos que presentan un mayor porcentaje de personas con ERC son Bogotá D.C., Antioquia, Risaralda, Caldas, Santander, Atlántico, Boyacá y Quindío. Las zonas con valores más altos corresponden a la región central de país, especialmente en los departamentos ubicados en el eje cafetero.

Trabajo futuro

El resultado positivo obtenido en el presente trabajo plantea la posibilidad de utilizar clasificadores con redes neuronales para realizar el pronóstico del riesgo de desarrollar otras enfermedades, especialmente aquellas que causan alta mortalidad entre la población, y aquellas que generan altos costos para el sistema de salud, como cáncer, enfermedades cardiovasculares y diabetes. Es posible ampliar el conjunto de variables agregando otros factores de riesgo como los antecedentes familiares. Los antecedentes familiares han demostrado tener un gran impacto como factor de riesgo, especialmente en enfermedades de tipo cardiovascular (Rodríguez et al., 2014). Esta información se podría obtener a partir de las bases de datos de afiliaciones a salud, en donde se indica el parentesco entre el afiliado y sus beneficiarios.

Otros datos de interés para el entrenamiento de los clasificadores son los resultados de las pruebas de laboratorio y los medicamentos suministrados al paciente. Para obtener estas variables es necesario integrar otras bases de datos en las que se encuentra información para grupos de población con patologías específicas. También se puede agregar información relacionada con defunciones para realizar análisis conjuntos de mortalidad y morbilidad. Estos análisis pueden aportar información relevante sobre la relación entre el historial médico del paciente y su causa de muerte.

Los clasificadores entrenados para pronosticar enfermedades específicas se pueden integrar para producir un sistema de medición del riesgo en forma de API o servicio web. A este sistema podrían acceder las entidades prestadoras de servicios de salud para cargar la información de sus pacientes, o incluso los mismos ciudadanos, con el fin de identificar el nivel de riesgo individual para desarrollar este conjunto de enfermedades.

Bibliografía

Abrahams, S., Hafner, D., Erwitt, E. y Scarpinelli, A. (2016). *TensorFlow for Machine Intelligence: A Hands-on Introduction to Learning Algorithms.* Bleeding Edge Press.

Al Imran, A., Amin, M. N. y Johora, F. T. (2018). Classification of Chronic Kidney Disease using Logistic Regression, Feedforward Neural Network and Wide & Deep Learning. *2018 International Conference on Innovation in Engineering and Technology*, 1-6.

Bishop, C. M. (1995). Regularization and complexity control in feed-forward networks. *Proceedings International Conference on Artificial Neural Networks*, 141–148.

Breiman, L. (1994). Bagging predictors. *Machine Learning, 24*(2), 123-140.

Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5-32.

Cao, Y., Hu, Z. D., Liu, X. F., Deng, A. M. y Hu, C. J. (2013). An MLP classifier for prediction of HBV-induced liver cirrhosis using routinely available clinical parameters. *Disease markers, 35*(6), 653-660.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C. y Wirth, R. (2000). *CRISP-DM 1.0: Step-by-step data mining guide.*

Cortes, C. y Vapnik, V. (1995). Support-vector networks. *Machine learning, 20*(3), 273-297.

DANE. (11 de mayo de 2019). *Proyecciones de población.* Obtenido de https://www.dane.gov.co/index.php/estadisticas-por-tema/demografia-y-poblacion/proyecciones-de-poblacion

Dessai, I. S. (2013). Intelligent heart disease prediction system using probabilistic neural network. *International Journal on Advanced Computer Theory and Engineering (IJACTE), 2*(3), 2319-2526.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861–874.

Flores, J. C., Alvo, M., Borja, H., Morales, J., Vega, J., Zúñiga, C. y Münzenmayer, J. (2009). Enfermedad renal crónica: Clasificación, identificación, manejo y complicaciones. *Revista médica de Chile, 137*(1), 137-177.

Fondo Colombiano de Enfermedades de Alto Costo. (2014). *Enfermedad Renal Crónica ERC*. Obtenido de http://www.cuentadealtocosto.org/index.php/patologias/9-patologias/35-enfermedad-renal-cronica-erc

Fondo Colombiano de Enfermedades de Alto Costo. (2017). *Situación de la enfermedad renal crónica, la hipertensión arterial y la diabetes mellitus en Colombia.* Bogotá.

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep Learning.* MIT Press.

Gulli, A. y Pal, S. (2017). *Deep Learning with Keras.* Packt Publishing Ltd.

Hinton, G. (2012). Neural networks for machine learning. *Coursera, video lectures*.

Hinton, G. E., Osindero, S. y Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 1527–1554.

Hore, S., Chatterjee, S., Shaw, R. K., Dey, N. y Virmani, J. (2018). Detection of chronic kidney disease: A NN-GA-based approach. *Nature Inspired Computing*, 109-115.

Hornik, K., Stinchcombe, M. y White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*, 359-366.

IBM Corporation. (2012). *Manual CRISP-DM de IBM SPSS.*

James, G., Witten, D., Hastie, T. y Tibshirani, R. (2013). *An Introduction to Statistical Leaerning with Applications in R.* New York: Springer.

Jarrett, K., Kavukcuoglu, K. y LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? *2009 IEEE 12th international conference on computer vision*, 2146-2153.

Kingma, D. P. y Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kriplani, H., Patel, B. y Roy, S. (2019). Prediction of Chronic Kidney Diseases Using Deep Artificial Neural Network Technique. *Computer Aided Intervention and Diagnostics in Clinical and Medical Images*, 179-187.

Kumar, K. y Abhishek, B. (2009). Artificial neural networks for diagnosis of kidney stones disease. *Information Technology and Computer Science, 1*, 41-48.

Magnin, B., Mesrob, L., Kinkingnéhun, S., Pélégrini-Issac, M., Colliot, O., Sarazin, M. y Benali, H. (2009). Support vector machine-based classification of Alzheimer's disease from whole-brain anatomical MRI. *Neuroradiology, 51*(2), 73-83.

Maltarollo, V. G., Honório, K. M. y da Silva, A. F. (2013). Applications of artificial neural networks in chemical problems. *Artificial neural networks-architectures and applications*, 203-223.

McCulloch, W. S. y Pitts, W. (1943). A logical calculus of ideas immanent in nervous. *Bulletin of Mathematical Biophysics, 5*, 115–133.

Ministerio de Salud y Protección Social. (2016). *Guía de Práctica Clínica para el diagnóstico y tratamiento de la Enfermedad Renal Crónica.* Bogotá.

Ministerio de Salud y Protección Social. (13 de mayo de 2019). *Misión Institucional*. Obtenido de https://www.minsalud.gov.co/Ministerio/Institucional/Paginas/mision-vision-principios.aspx

Ministerio de Salud y Protección Social. (13 de mayo de 2019). *Resolución 3374 de 2000.* Obtenido de https://www.minsalud.gov.co/sites/rid/Lists/BibliotecaDigital/RIDE/DE/DIJ/Resoluci%C3%B3n_3374_de_2000.pdf

Ministerio de Salud y Protección Social. (13 de mayo de 2019). *Sistema de Información de Prestaciones de Salud*. Obtenido de https://www.minsalud.gov.co/proteccionsocial/Paginas/rips.aspx

Ministerio de Salud y Protección Social. (28 de abril de 2019). *Sistema Integrado de Información de la Protección Social*. Obtenido de https://www.sispro.gov.co/Pages/Home.aspx

Nielsen, M. (2015). *Neural Networks and Deep Learning.* San Francisco, CA, USA: Determination press.

Organización Panamericana de la Salud. (2015). *La OPS/OMS y la Sociedad Latinoamericana de Nefrología llaman a prevenir la enfermedad renal y a mejorar el acceso al tratamiento*. Obtenido de https://www.paho.org/hq/index.php?option=com_content&view=article&id=10542:2015-opsoms-sociedad-latinoamericana-nefrologia-enfermedad-renal-mejorar-tratamiento&Itemid=1926

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

Ren, Y., Fei, H., Liang, X., Ji, D. y Cheng, M. (2019). A hybrid neural network model for predicting kidney disease in hypertension patients based on electronic health records. *BMC Medical Informatics and Decision Making, 19*(2), 51.

Rodríguez, L., Díaz, M. E., Ruiz, V., Hernández, H., Herrera, V. y Montero, M. (2014). Factores de riesgo cardiovascular y su relación con la hipertensión arterial en adolescentes. *Revista Cubana de Medicina, 53*(1), 25-36.

Rumelhart, D., Hinton, G. y Williams, R. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533–536.

Samuel, A. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Developmen, 3*(3), 210-29.

Soldevila, A. (2017). *Análisis de la progresión de la enfermedad renal crónica avanzada mediante técnicas de aprendizaje máquina.* Valencia.

Srivastava, N. (2013). Improving neural networks with dropout. *Universidad de Toronto*.

Tretyakov, K. (2004). Machine learning techniques in spam filtering. *Data Mining Problem-oriented Seminar, MTAT, 3*(177), 60-79.

Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data.* New York: Springer-Verlag.

West, D. y West, V. (2000). Improving diagnostic accuracy using a hierarchical neural network to model decision subtasks. *International journal of medical informatics, 57*(1), 41-55.

Xiao, J., Ding, R., Xu, X., Guan, H., Feng, X., Sun, T. y Ye, Z. (2019). Comparison and development of machine learning tools in the prediction of chronic kidney disease progression. *Journal of translational medicine, 17*(1), 119.

Yang, J. y Yang, G. (2018). Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer. *Algorithms, 11*(3), 28.

Yu, W., Liu, T., Valdez, R., Gwinn, M. y Khoury, M. J. (2010). Application of support vector Machine modeling for prediction of common diseases: the case of diabetes and pre-diabetes. *BMC medical informatics and decision making, 10*(1), 16.

Zhang, H., Hung, C. L., Chu, W. C., Chiu, P. F. y Tang, C. Y. (2018). Chronic Kidney Disease Survival Prediction with Artificial Neural Networks. *2018 IEEE International Conference on Bioinformatics and Biomedicine*, 1351-1356.

Zhou, Z. H., Jiang, Y., Yang, Y. B. y Chen, S. F. (2002). Lung cancer cell identification based on artificial neural network ensembles. *Artificial Intelligence in Medicine, 24*(1), 25-36.

Anexo 1

Código fuente

1. Limpieza y preparación de los datos

1.1 Carga librerias

```
#importa librerias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
from time import time
import itertools
```

1.2 Carga tabla de Diagnosticos

```
#Carga datos de diagnosticos a dataframe
dfDiagnosticos = pd.read_csv("Diagnosticos.csv",sep=",")
dfDiagnosticos.head()
```

1.3 Realiza pivot por columnas

```
#Realiza pivot de columnas
dfDiagnosticos = dfDiagnosticos.pivot(index='PersonaID',
columns='CodigoDiagnostico', values='NumeroAtenciones')

#Reemplaza NA con 0
dfDiagnosticos = dfDiagnosticos.fillna(0)

#Muestra datos
dfDiagnosticos.head()
```

1.4 Carga archivo de personas

```
#Carga datos de personas a dataframe
dfPersonas = pd.read_csv("Personas.csv")
dfPersonas.head()
```

1.5 Transforma variables categóricas

```
#Transforma variables categóricas
dfPersonas = pd.get_dummies(dfPersonas, columns=['Sexo'], prefix = ['Sexo'])
dfPersonas = pd.get_dummies(dfPersonas, columns=['Etnia'], prefix = ['Etnia'])
dfPersonas = pd.get_dummies(dfPersonas, columns=['Departamento'], prefix = ['Departamento'])
dfPersonas = dfPersonas.drop(['Etnia_NINGUNA'], axis=1)
dfPersonas = dfPersonas.drop(['Departamento_NO_DEFINIDO'], axis=1)
dfPersonas.head()
```

1.6 Realiza join entre los dos archivos

```
#Realiza join entre los dos dataframes
dfConsolidado = pd.merge(dfDiagnosticos, dfPersonas, on='PersonaID', how='left')
dfConsolidado.head()
```

1.7 Genera archivo para modelo

```
#dfConsolidado.to_csv('DatosERC.csv',sep=',',index=False)
```

2. Modelado de la red neuronal

2.1 Selección de features

```
#Reemplaza NA con 0
dfDatosERC = dfConsolidado.fillna(0)

#Retira columna de ID
dfDatosERC = dfDatosERC.drop(['PersonaID'], axis=1)

#Muestra primeras 5 lineas
dfDatosERC.head()
```

2.2 Separación de features (x) y etiquetas (y)

```
#Separa datos x(features) vs y(clase)
y = dfDatosERC['ERC']
x = dfDatosERC.drop(['ERC'], axis=1)

#Normaliza datos en x
```

```
x = (x - x.min()) / (x.max() - x.min())
```

2.3 Conjuntos de datos de entrenamiento y pruebas

```
#Separa datos de entrenamiento y pruebas
entrenamientoX, pruebasX, entrenamientoY, pruebasY = train_test_split(x,y,
test_size= 0.30, random_state=2)
entrenamientoX.head()
```

2.4 Diseño del modelo de la red neuronal

```
#Definición del modelo
model = Sequential()
model.add(Dense(units=500, activation='sigmoid', input_dim=7492))
model.add(Dense(units=100, activation='sigmoid'))
model.add(Dense(units=50, activation='sigmoid'))
model.add(Dense(units=1, activation='sigmoid'))

#compilación del modelo
model.compile(loss='binary_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#Resumen del modelo
model.summary()
```

2.5 Entrenamiento de la red neuronal

```
#entrenamiento del modelo
history_01=model.fit(entrenamientoX, entrenamientoY, validation_split=0.2,
epochs=10, batch_size=1000)
```

2.6 Pruebas del modelo con datos de test

```
#Realiza prediccion en conjunto pruebas
score = model.evaluate(pruebasX, pruebasY)

#Obtiene el accuracy del modelo final
print("Valor de accuracy en conjunto de datos de test: ",score[1])
```

2.7 Función de activación (sigmoid vs relu)

```
#Definición del modelo
model = Sequential()
model.add(Dense(units=500, activation='relu', input_dim=7492))
model.add(Dense(units=100, activation='relu'))
```

```
model.add(Dense(units=50, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

#compilación del modelo
model.compile(loss='binary_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#entrenamiento del modelo
history_02=model.fit(entrenamientoX, entrenamientoY, validation_split=0.2,
epochs=10, batch_size=1000)


#Realiza prediccion en conjunto pruebas
score = model.evaluate(pruebasX, pruebasY)

#Obtiene el accuracy del modelo final
print("Valor de accuracy en conjunto de datos de test: ",score[1])
```

2.8 Visualización de resultados

```
#Gráfica del loss y el accuracy

def plot_acc(history, title="Model Accuracy"):
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title(title)
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

def plot_loss(history, title="Model Loss"):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(title)
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper right')
    plt.show()

def plot_compare_losses(history1, history2, name1="Red 1",
                        name2="Red 2", title="Graph title"):
    plt.plot(history1.history['loss'], color="green")
    plt.plot(history1.history['val_loss'], 'r--', color="green")
    plt.plot(history2.history['loss'], color="blue")
    plt.plot(history2.history['val_loss'], 'r--', color="blue")
    plt.title(title)
```

```
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train ' + name1, 'Val ' + name1,
                'Train ' + name2, 'Val ' + name2],
               loc='upper right')
    plt.show()

def plot_compare_accs(history1, history2, name1="Red 1",
                      name2="Red 2", title="Graph title"):
    plt.plot(history1.history['acc'], color="green")
    plt.plot(history1.history['val_acc'], 'r--', color="green")
    plt.plot(history2.history['acc'], color="blue")
    plt.plot(history2.history['val_acc'], 'r--', color="blue")
    plt.title(title)
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train ' + name1, 'Val ' + name1,
                'Train ' + name2, 'Val ' + name2],
               loc='lower right')
    plt.show()

plot_compare_losses(history_01, history_02, name1="Sigmoid",
                      name2="Relu", title="Comparación loss con diferentes
funciones de activacion")
plot_compare_accs(history_01, history_02, name1="Sigmoid",
                      name2="Relu", title="Comparación accuracy con
diferentes funciones de activacion")
```

2.9 Algoritmo de entrenamiento (sgd vs adam)

```
#Definición del modelo
model = Sequential()
model.add(Dense(units=500, activation='relu', input_dim=7492))
model.add(Dense(units=100, activation='relu'))
model.add(Dense(units=50, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

#compilación del modelo
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

#entrenamiento del modelo
history_03=model.fit(entrenamientoX, entrenamientoY, validation_split=0.2,
epochs=10, batch_size=1000)
```

```
#Realiza prediccion en conjunto pruebas
score = model.evaluate(pruebasX, pruebasY)

#Obtiene el accuracy del modelo final
print("Valor de accuracy en conjunto de datos de test: ",score[1])

plot_compare_losses(history_02, history_03, name1="SGD",
                    name2="Adam", title="Comparación loss con diferentes
optimizadores")
plot_compare_accs(history_02, history_03, name1="SGD",
                    name2="Adam", title="Comparación accuracy con
diferentes optimizadores")
```

2.10 Regularización (dropout)

```
#Definición del modelo
model = Sequential()
model.add(Dense(units=500, activation='relu', input_dim=7492))
model.add(Dropout(0.5))
model.add(Dense(units=100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=50, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1, activation='sigmoid'))

#compilación del modelo
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

#entrenamiento del modelo
history_04=model.fit(entrenamientoX, entrenamientoY, validation_split=0.2,
epochs=10, batch_size=1000)

#Realiza prediccion en conjunto pruebas
score = model.evaluate(pruebasX, pruebasY)

#Obtiene el accuracy del modelo final
print("Valor de accuracy en conjunto de datos de test: ",score[1])

plot_compare_losses(history_03, history_04, name1="Sin reg",
                    name2="Con reg", title="Comparación loss con
regularización")
plot_compare_accs(history_03, history_04, name1="Sin reg",
                    name2="Con reg", title="Comparación accuracy con
regularización")
```

2.11 Modelo final

```
#Entrena modelo final
tiempo_inicial = time()

#Definición del modelo
model_final = Sequential()
model_final.add(Dense(units=500, activation='relu', input_dim=7492))
model_final.add(Dropout(0.5))
model_final.add(Dense(units=100, activation='relu'))
model_final.add(Dropout(0.5))
model_final.add(Dense(units=50, activation='relu'))
model_final.add(Dropout(0.5))
model_final.add(Dense(units=1, activation='sigmoid'))

#compilación del modelo
model_final.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

#Agrega early stopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

#entrenamiento del modelo
history_final=model_final.fit(entrenamientoX,                entrenamientoY,
validation_split=0.2, epochs=3, batch_size=1000, callbacks=[es])

tiempo_final = time()
print   ('El   tiempo   de   ejecucion   fue:',   tiempo_final-tiempo_inicial,
"segundos")


#Realiza prediccion en conjunto pruebas
score = model_final.evaluate(pruebasX, pruebasY)

#Obtiene el accuracy del modelo final
print("Valor de accuracy en conjunto de datos de test: ",score[1])


plot_loss(history_final, title="Loss de la red neuronal")
plot_acc(history_final, title="Accuracy de la red neuronal")

#Guarda modelo final
model_final.save('modelo_Red_Neuronal_ERC.h5')
```

3. Evaluación del modelo

3.1 Matriz de confusión

```
#Obtiene valores predichos para el conjunto de pruebas
scores = model_final.predict(pruebasX, verbose=0)
prediccion = np.round(scores).astype(int)
prediccion

#Genera matriz de confusión
cf = confusion_matrix(prediccion, pruebasY)
cf

#Grafica matriz de confusión
plt.imshow(cf,cmap=plt.cm.Blues,interpolation='nearest')
plt.colorbar()
plt.title('Matriz de confusión')
plt.xlabel('Predicho')
plt.ylabel('Actual')
tick_marks = np.arange(len(set(pruebasY))) # length of classes
class_labels = ['0','1']
tick_marks
plt.xticks(tick_marks,class_labels)
plt.yticks(tick_marks,class_labels)
# plotting text value inside cells
thresh = cf.max() / 2.
for i,j in itertools.product(range(cf.shape[0]),range(cf.shape[1])):

plt.text(j,i,format(cf[i,j],'d'),horizontalalignment='center',color='white'
if cf[i,j] >thresh else 'black')
plt.show();
```

3.2 Precision, Recall y F1-score

```
#Obtiene metricas
tp=cf[1,1]
tn=cf[0,0]
fp=cf[1,0]
fn=cf[0,1]
print("La metrica de sensibilidad para el modelo es
de:",round(tp/(tp+fn),2))
print("La metrica de especificidad para el modelo es
de:",round(tn/(tn+fp),2))
pre=round(tp/(tp+fp),2)
print("La metrica de precision para el modelo es de:",pre)
```

```
rec=round(tp/(tp+fn),2)
print("La metrica de recall para el modelo es de:",rec)
print("La metrica de f1-score para el modelo es de:",round(
(2*pre*rec)/(pre+rec),2) )
```

3.3 Curva ROC/AUC

```
#Obtiene AUC
auc=roc_auc_score(pruebasY, scores)
print("El AUC es de:",auc)

#Genera ROC
fpr, tpr, thresholds = roc_curve(pruebasY, scores)

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='Curva ROC (AUC = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('ROC/AUC')
plt.legend(loc="lower right")
plt.show()
```

4. Modelo con SVM

4.1 Entrenamiento con SVM

```
#Realiza entrenamiento con SVM
#from sklearn.svm import SVC
clfSVM = SVC(gamma='auto')
clfSVM.fit(entrenamientoX, entrenamientoY)
```

4.2 Pruebas del modelo con SVM

```
#Realiza prediccion en conjunto pruebas con SVM
scores=clfSVM.predict(pruebasX)
prediccionSVM=np.round(scores).astype(int)
prediccionSVM
```

4.3 Evaluación de resultados con SVM

```
#Genera matriz de confusión
cf = confusion_matrix(prediccionSVM, pruebasY)
cf


#Grafica matriz de confusión
plt.imshow(cf,cmap=plt.cm.Blues,interpolation='nearest')
plt.colorbar()
plt.title('Matriz de confusión')
plt.xlabel('Predicho')
plt.ylabel('Actual')
tick_marks = np.arange(len(set(pruebasY))) # length of classes
class_labels = ['0','1']
tick_marks
plt.xticks(tick_marks,class_labels)
plt.yticks(tick_marks,class_labels)
# plotting text value inside cells
thresh = cf.max() / 2.
for i,j in itertools.product(range(cf.shape[0]),range(cf.shape[1])):

plt.text(j,i,format(cf[i,j],'d'),horizontalalignment='center',color='white'
if cf[i,j] >thresh else 'black')
plt.show();
```

4.4 Métricas de evaluación

```
#Obtiene metricas
tp=cf[1,1]
tn=cf[0,0]
fp=cf[1,0]
fn=cf[0,1]
print("La metrica de accuracy para el modelo es
de:",round((tp+tn)/(tp+tn+fn+fp),2))
print("La metrica de sensibilidad para el modelo es
de:",round(tp/(tp+fn),2))
print("La metrica de especificidad para el modelo es
de:",round(tn/(tn+fp),2))
pre=round(tp/(tp+fp),2)
print("La metrica de precision para el modelo es de:",pre)
rec=round(tp/(tp+fn),2)
print("La metrica de recall para el modelo es de:",rec)
print("La metrica de f1-score para el modelo es de:",round(
(2*pre*rec)/(pre+rec),2) )
```

4.5 Curva ROC/AUC

```
#Obtiene AUC
auc=roc_auc_score(pruebasY, scores)
print("El AUC es de:",auc)

#Genera ROC
fpr, tpr, thresholds = roc_curve(pruebasY, scores)

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='Curva ROC (AUC = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('ROC/AUC')
plt.legend(loc="lower right")
plt.show()
```

5. Modelo con Random Forest

5.1 Entrenamiento con Random Forest

```
#Realiza entrenamiento con Random Forest
clfRF = RandomForestClassifier(n_estimators=10)
clfRF.fit(entrenamientoX, entrenamientoY)

#Muestra las features mas importantes
importancia = pd.DataFrame(columns=('variable', 'importancia'))
cont=0
for i in pruebasX.columns:
    importancia.loc[len(importancia)]=[i,clfRF.feature_importances_[cont]]
    cont=cont+1
importancia=importancia.sort_values(by=['importancia'], ascending=[False])
print(importancia[:20])
```

5.2 Pruebas del modelo con Random Forest

```
#Realiza prediccion en conjunto pruebas con SVM
scores=clfRF.predict(pruebasX)
prediccionRF=np.round(scores).astype(int)
```

```
prediccionRF
```

5.3 Evaluación de resultados con Random Forest

```
#Genera matriz de confusión
cf = confusion_matrix(prediccionRF, pruebasY)
cf


#Grafica matriz de confusión
plt.imshow(cf,cmap=plt.cm.Blues,interpolation='nearest')
plt.colorbar()
plt.title('Matriz de confusión')
plt.xlabel('Predicho')
plt.ylabel('Actual')
tick_marks = np.arange(len(set(pruebasY))) # length of classes
class_labels = ['0','1']
tick_marks
plt.xticks(tick_marks,class_labels)
plt.yticks(tick_marks,class_labels)
# plotting text value inside cells
thresh = cf.max() / 2.
for i,j in itertools.product(range(cf.shape[0]),range(cf.shape[1])):

plt.text(j,i,format(cf[i,j],'d'),horizontalalignment='center',color='white'
if cf[i,j] >thresh else 'black')
plt.show();
```

5.4 Métricas de evaluación

```
#Obtiene metricas
tp=cf[1,1]
tn=cf[0,0]
fp=cf[1,0]
fn=cf[0,1]
print("La metrica de accuracy para el modelo es
de:",round((tp+tn)/(tp+tn+fn+fp),2))
print("La metrica de sensibilidad para el modelo es
de:",round(tp/(tp+fn),2))
print("La metrica de especificidad para el modelo es
de:",round(tn/(tn+fp),2))
pre=round(tp/(tp+fp),2)
print("La metrica de precision para el modelo es de:",pre)
rec=round(tp/(tp+fn),2)
print("La metrica de recall para el modelo es de:",rec)
print("La metrica de f1-score para el modelo es de:",round(
(2*pre*rec)/(pre+rec),2) )
```

5.5 Curva ROC/AUC

```
#Obtiene AUC
auc=roc_auc_score(pruebasY, scores)
print("El AUC es de:",auc)

#Genera ROC
fpr, tpr, thresholds = roc_curve(pruebasY, scores)

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='Curva ROC (AUC = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('ROC/AUC')
plt.legend(loc="lower right")
plt.show()
```

6. Implementación del modelo

6.1 Carga librerias

```
#importa librerias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import normalize
from time import time
import gc
from os import remove

import rarfile
```

6.2 Carga modelo entrenado

```
#Carga modelo final
model_final = keras.models.load_model('modelo_Red_Neuronal_ERC.h5')
```

6.3 Aplica el modelo entrenado a los datos de cada archivo

```
#Mide tiempo inicial
tiempo_inicial = time()

#Define número de archivos
numArchivos=392

#Recorre cada archivo
for i in range(numArchivos):

    #Obtiene nombre del archivo
    if(i<10):
        nomArchivo="ExtraccionRIPSConsolidada_00"+str(i)+".csv"
        nomArchivoRAR="ExtraccionRIPSConsolidada_00"+str(i)+".rar"
    elif(i<100):
        nomArchivo="ExtraccionRIPSConsolidada_0"+str(i)+".csv"
        nomArchivoRAR="ExtraccionRIPSConsolidada_0"+str(i)+".rar"
    else:
        nomArchivo="ExtraccionRIPSConsolidada_"+str(i)+".csv"
        nomArchivoRAR="ExtraccionRIPSConsolidada_"+str(i)+".rar"

    #Extrae el archivo
    rar = rarfile.RarFile(nomArchivoRAR)
    rar.extractall()

    #Carga datos del archivo consolidado a dataframe
    dfDatosERC = pd.read_csv(nomArchivo)
    dfPersonas = dfDatosERC['PersonaID']
    dfDatosERC = dfDatosERC.drop(['PersonaID'], axis=1)
    #dfDatosERC.head(10)

    #Aplica modelo a los datos cargados
    scores = model_final.predict(dfDatosERC)
    prediccion = np.round(scores).astype(int)

    #Resumen de Personas identificadas
    print("En el archivo ",i," se encontraron ",prediccion.sum(),
          "personas con ERC de un total de ", len(prediccion),
          ",                    correspondiente                    al",
round(prediccion.sum()/len(prediccion)*100,2), "%")
```

```
    #Exporta la personas identificadas con ERC junto con su respectiva
probabilidad
    archivoPronostico = open('PronosticoPersonas.csv','a')
    cont=0
    for i in dfPersonas:
        if(prediccion[cont]==1):
            archivoPronostico.write(str(i)+','+str(scores[cont])+'\n')
        cont=cont+1
    archivoPronostico.close()

    #Limpia memoria
    del [[dfDatosERC,dfPersonas,scores,prediccion]]
    gc.collect()
    dfDatosERC=pd.DataFrame()
    dfPersonas=pd.DataFrame()

    #Elimina archivo descomprimido
    remove(nomArchivo)

tiempo_final = time()
print ('El tiempo de ejecucion fue:', tiempo_final-tiempo_inicial,
"segundos")
```

Anexo 2

Artículo de investigación

Clasificador con redes neuronales para el pronóstico de la enfermedad renal crónica en la población colombiana

Gabriel Ricardo Vásquez Morales

Universidad Internacional de la Rioja, Logroño (España)

24 de Julio de 2019

Resumen

Este trabajo tiene como objetivo la creación de un clasificador basado en redes neuronales para pronosticar si una persona está en riesgo de desarrollar enfermedad renal crónica (ERC). El modelo se entrenó con los datos demográficos y la información de las atenciones médicas de dos grupos poblacionales: por una parte, personas diagnosticadas con ERC en Colombia durante el año 2018, y por otra, una muestra de personas sin diagnóstico de esta enfermedad. Una vez entrenado el modelo y aplicadas las métricas de evaluación para algoritmos de clasificación, el modelo alcanzó 95% de exactitud en el conjunto de datos de prueba, siendo por lo tanto viable su aplicación para el pronóstico de la enfermedad. Tras su implementación, se identificaron 3.494.516 personas en riesgo de desarrollar ERC en Colombia, es decir un 7% del total de la población.

Palabras Clave

clasificador, enfermedad renal crónica, pronóstico, red neuronal.

I. Introducción

La enfermedad renal crónica, también conocida por su sigla ERC, consiste en una disminución progresiva e irreversible de la función renal, que trae como consecuencia la incapacidad de los riñones para eliminar los desechos a través de la orina [1] [2]. Esta enfermedad afecta aproximadamente al 10% de la población mundial, siendo una patología silenciosa que no presenta síntomas hasta que ya se encuentra en un estado avanzado [3]. El tratamiento para la ERC además de ser altamente invasivo e impactar negativamente en la calidad de vida del paciente, también es bastante costoso para los sistemas de salud [4]. En Colombia, hasta el año 2017 había 1.406.364 personas con Enfermedad Renal Crónica, es decir una prevalencia de 2,9 casos por cada 100 habitantes [4]. La Figura 1 muestra el incremento en la prevalencia de personas con ERC en los últimos años.

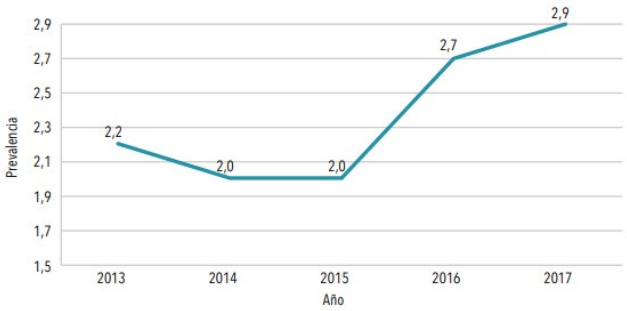


Fig. 1. Prevalencia de la ERC en Colombia. La cifra aumentó de 2 personas por cada 100 habitantes, en 2015, a 2.9 personas para el año 2017. Fuente: Fondo Colombiano de Enfermedades de Alto Costo [4].

En la actualidad las redes neuronales [5] [6] [7], y en general los algoritmos de aprendizaje automático [8] [9] han comenzado a utilizarse de forma exitosa como herramientas de apoyo para el diagnóstico y detección temprana de enfermedades [10]. Estos algoritmos se entrenan con grandes volúmenes de datos, que incluyen imágenes diagnósticas, pruebas de laboratorio e historias clínicas. En el presente trabajo se propuso el uso de una red neuronal para el pronóstico de la enfermedad renal crónica en la población colombiana. Para el entrenamiento del clasificador se tomó la información de dos grupos poblacionales, por una parte, 20.000 pacientes a quienes se les diagnosticó ERC durante el año 2018, y, por otra parte, una muestra igual de personas sanas. Para ambos grupos de población se recopiló la información demográfica, compuesta por las variables de sexo, edad, lugar de residencia y etnia, junto con el historial de diagnósticos de enfermedades, obtenidos de la base de datos del RIPS [11] [12] [13] (Registro individual de prestación de servicios en salud) del ministerio de salud y protección social de Colombia [14]. Teniendo como base este conjunto de datos se realizó el entrenamiento y optimización de la red neuronal, así como de otros dos modelos de aprendizaje automático, uno con máquina de soporte de vectores (SVM) y otro con Random Forest.

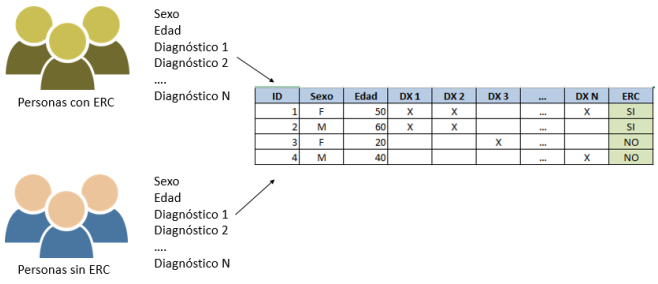


Fig. 2. Conjuntos de datos para el entrenamiento del modelo. A partir de la información demográfica y médica de las personas enfermas y de las personas sanas se creó el conjunto de datos compuesto por las variables de entrada y la etiqueta que identifica si la persona tiene o no ERC.

Finalmente se aplicaron métricas de evaluación a los clasificadores entrenados con los tres algoritmos, y se implementó el modelo de redes neuronales sobre el conjunto de datos disponibles para la totalidad de la población colombiana, con el propósito de pronosticar nuevos casos de ERC.

II. Estado del Arte

Uno de los campos de aplicación de la inteligencia artificial en la actualidad es el sector salud. Los algoritmos de aprendizaje automático han sido ampliamente usados en tareas de predicción y clasificación de enfermedades. Algunos algoritmos como SVM, Random Forest y redes neuronales, han sido utilizados para predecir y clasificar pacientes con diabetes [15], Alzheimer [16], enfermedades del corazón [17], cáncer [18] y cirrosis hepática [19], entre otras. Las redes neuronales también se han utilizado para la detección de ERC y otras patologías relacionadas con el sistema urinario. Algunos modelos han permitido detectar cálculos renales [20] a partir de los resultados de exámenes de laboratorio como prueba de creatinina, ácido úrico, glucosa, linfocitos y otros componentes sanguíneos. Otros trabajos se han enfocado en predecir la supervivencia de pacientes con ERC [21]. Esta es una tarea compleja debido a la cantidad de variables a tener en cuenta para determinar el tiempo de vida restante en personas que se encuentran en una etapa avanzada de la enfermedad. También se han aplicado redes neuronales y otras técnicas de aprendizaje automático para identificar la etapa de la enfermedad renal crónica en la que se encuentra un paciente [22] [23] [24].

En otros estudios se propone un modelo de redes neuronales para detectar ERC a partir de datos de laboratorio de los pacientes [25] [26], así como realizar comparaciones con otros modelos de aprendizaje automático. Los resultados de las métricas de evaluación han demostrado la efectividad de los modelos entrenados con redes neuronales, obteniendo valores hasta del 97% de exactitud [27]. Un estudio que se acerca bastante al objetivo del presente trabajo es el de [28], en el que se desarrolló una red neuronal híbrida con el fin de pronosticar ERC en pacientes con hipertensión arterial a partir de sus registros médicos. Los resultados del experimento demostraron que la red neuronal fue capaz de predecir correctamente los casos de ERC con una exactitud del 89.7%.

En general todos estos estudios utilizan como variables de entrada para el entrenamiento de los modelos la información de pruebas de laboratorio y se dispone de un conjunto relativamente pequeño de pacientes. En este trabajo se cuenta con un conjunto de datos de entrenamiento correspondiente a 40.000 personas, y para la implementación del modelo se utilizó la información de atenciones en salud realizadas a 39.277.086 personas durante los años 2009 a 2018. Estas variables no incluyen información de resultados de pruebas de laboratorio, pero si incluye información demográfica, como sexo, edad, etnia y lugar de residencia, así como el historial de las patologías que le han sido diagnosticadas a la persona, codificadas de acuerdo a la Clasificación Internacional de Enfermedades (CIE).

III. Objetivos y Metodología

El objetivo general del trabajo consiste en construir una red neuronal que permita pronosticar el riesgo de desarrollar enfermedad renal crónica a partir de la información reportada en los Registros Individuales de Prestación de Servicios (RIPS), obteniendo una métrica de exactitud superior al 90%.

Los objetivos específicos son:

1. Identificar el algoritmo de aprendizaje automático con mayor capacidad para el pronóstico de la enfermedad renal crónica a partir del conjunto de datos demográficos y de atenciones médicas obtenidos de la base de datos RIPS.

2. Implementar el modelo de redes neuronales para el pronóstico de la ERC en la población colombiana haciendo uso de los registros médicos disponibles en la base de datos de RIPS.

3. Analizar el perfil demográfico de los pacientes a quienes la red neuronal pronostica con alto riesgo de desarrollar ERC.

Para el desarrollo del trabajo se empleó la metodología CRISP-DM, considerada un estándar para el ciclo de vida de proyectos de analítica de datos. En la Figura 3 se presentan las seis etapas que se llevan a cabo con la metodología CRISP-DM: entendimiento del negocio, entendimiento de los datos, preparación de los datos, modelado, evaluación del modelo, e implementación del modelo [29][30].

Fig. 3. Etapas de la metodología CRISP-DM. Fuente: Elaboración propia basada en [30].

IV. Contribución

A continuación, se describen los procesos realizados para la preparación y transformación de los datos, el entrenamiento, optimización y evaluación del modelo con redes neuronales.

A. Limpieza y preparación del conjunto de datos

El conjunto de datos requerido para el entrenamiento de la red neuronal se obtuvo a partir de la base de datos de RIPS (Registro Individual de Prestación de Servicios de Salud) del ministerio de salud y protección social de Colombia. Esta base de datos contiene la información de las atenciones en salud practicadas a todas las personas afiliadas al sistema de salud del país desde el año 2009. Para obtener la muestra de individuos con ERC se identificaron las personas cuyos diagnósticos en la base de datos de RIPS correspondían con códigos dentro del subgrupo N18 de la Clasificación Internacional de Enfermedades (CIE). Este subgrupo corresponde a los diagnósticos de “Insuficiencia renal crónica”. Luego se seleccionaron aquellas personas cuya fecha de diagnóstico correspondía al año 2018. El número de personas obtenida en esta consulta fue de 203.015. Teniendo en cuenta que el entrenamiento del modelo puede llegar a ser costoso con un conjunto de datos tan grande se tomó una muestra aleatoria de 20.000 personas.

Para obtener el conjunto de personas del grupo de control se identificaron aquellas personas con atenciones durante año 2018 y se tomó una muestra aleatoria con el mismo número de elementos existente en el grupo de personas con ERC, es decir 20.000 individuos. En este grupo de personas se realizó un filtro para descartar aquellos a los que se les había diagnosticado ERC previamente. Una vez identificadas las personas, se integraron los dos conjuntos de datos y se agregaron las variables demográficas correspondientes a sexo, edad, etnia y departamento de residencia de la persona. También se agregaron todos los diagnósticos identificados en la base de datos de RIPS para cada persona, así como el número de atenciones médicas realizadas a causa de este diagnóstico. Sobre estos datos se efectuaron operaciones de selección de características, unión de tablas, transformación de filas a columnas, creación de variables categóricas, tratamiento de valores nulos y otras operaciones de limpieza requeridas para obtener el conjunto de datos final compuesto por 40.000 registros y 7.493 variables, incluyendo la clase o variable de salida.

B. *Definición de la topología de la red neuronal*

La red neuronal se diseñó con una topología compuesta por 5 capas de acuerdo con el diagrama de la Figura 4. La capa de entrada corresponde a las características o variables de entrada de la red, con 7.492 nodos o neuronas. Las siguientes 3 capas corresponden a las capas ocultas del modelo y contienen 500, 100 y 50 neuronas respectivamente. La última capa corresponde a la neurona que representa la única clase del problema de clasificación binario. El objetivo del entrenamiento es obtener los valores óptimos para los parámetros de la red, compuestos por los pesos (W) de cada capa y por los valores de bias [31]. Para todas las capas se aplicó inicialmente una función de activación de tipo sigmoide.

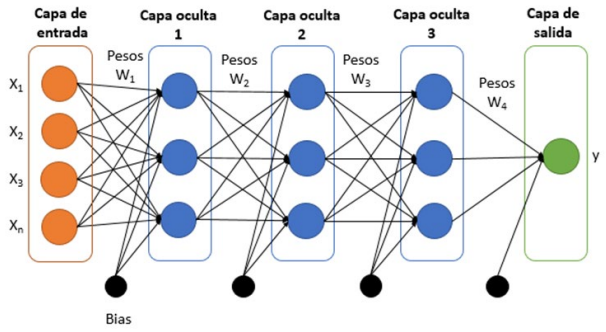


Fig. 4. Topología de la red neuronal con 5 capas. La primera capa corresponde a las variables de entrada del modelo, las siguientes tres son las capas ocultas y la última es la capa de salida.

Para la creación, evaluación e implementación de la red neuronal se utilizó el lenguaje de programación Python 3.7.0, en conjunto con la librería Keras 2.2.4 [32] y el framework TensorFlow 1.13.1 [33], sobre un entorno de desarrollo Jupyter Notebook 5.6.0. Una vez definido el modelo se realizó el entrenamiento haciendo uso de *backpropagation* [34] en conjunto con el algoritmo de *gradient descent*, aunque en la práctica se utiliza una variación de este algoritmo conocido como stochastic gradient descent (SGD) [35] que es menos costoso computacionalmente. La Figura 5 presenta el resumen de la red neuronal desarrollada en keras, junto con el número de parámetros a entrenar por cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_44 (Dense)             (None, 500)               3746500
_________________________________________________________________
dense_45 (Dense)             (None, 100)               50100
_________________________________________________________________
dense_46 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 51
=================================================================
Total params: 3,801,701
```

Fig. 5. Resumen del diseño de la red neuronal en Keras.

El número de parámetros en cada capa corresponde al número de nodos de la capa anterior, multiplicado por el número de nodos de la capa actual, más un valor de bias por cada nodo de la capa actual. Por ejemplo, para la primera capa oculta el número de parámetros esta dado por el número de nodos de la capa de entrada (7.492) multiplicado por el número de nodos de la primera capa oculta (500), más 500 valores de bias correspondientes a cada nodo de la capa actual. Esto suma un total de 3.746.500 parámetros para la primera capa oculta. Para las restantes capas el número de parámetros es de 50.100, 5.050 y 51, arrojando un valor total de 3.801.701 parámetros para toda la red.

Para el entrenamiento de la red se utilizó el 70% de los datos y se realizaron 10 iteraciones o épocas con un tamaño de lote de 1.000 elementos aplicando validación cruzada. Una vez entrenado el modelo se utilizó el 30% restante de los datos como conjunto de pruebas para identificar el desempeño obtenido con el modelo inicial. Como resultado se obtuvo un valor de exactitud igual a 49.91%. Teniendo en cuenta que este es valor muy bajo, fue necesario realizar ajustes en algunos hiperparámetros empleados en el entrenamiento del modelo. Esta optimización incluyó cambios en la función de activación, el algoritmo de entrenamiento, así como incluir funciones de normalización y early stopping.

C. *Optimización de hiperparámetros*

Durante el entrenamiento del modelo inicial de la red se obtuvo un valor de exactitud demasiado bajo, por lo tanto, se optó por cambiar la función de activación sigmoide por una función ReLU [36]. El valor de exactitud obtenido con este ajuste en el modelo fue del 59.96%. En la Figura 6 se compara el valor de la función de error, así como la exactitud obtenida para cada modelo.

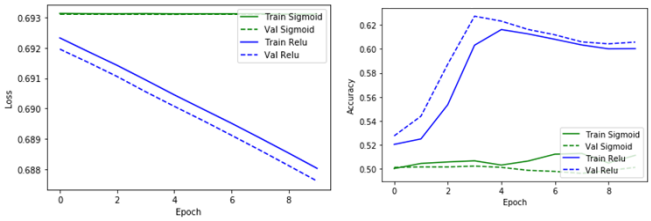


Fig. 6. Comparación del error y la exactitud obtenidos con diferentes funciones de activación. A la izquierda la función de error y a la derecha el valor de exactitud. En color verde los valores obtenidos con la función de activación sigmoide y en azul los obtenidos con la función de activación ReLU.

Como se puede observar en las gráficas anteriores, la función de activación ReLU permite a la red disminuir el valor del error y a la vez mejorar considerablemente la medida de exactitud, aunque este valor aún se encuentra por debajo del umbral deseado. Con el fin de mejorar el desempeño de la red se cambió el algoritmo de optimización de SGD a Adam [37] [38]. Esta optimización permitió obtener un valor de exactitud de 92.65%. Este valor mejora considerablemente la métrica obtenida anteriormente con SGD igual a 59.96%. También supera el umbral del 90% propuesto en los objetivos del proyecto.

En la Figura 7 se puede observar la disminución en la medida del error tanto en el conjunto de entrenamiento, como en el de validación utilizando optimización con Adam (en azul), frente a la obtenida anteriormente con SGD (en verde).

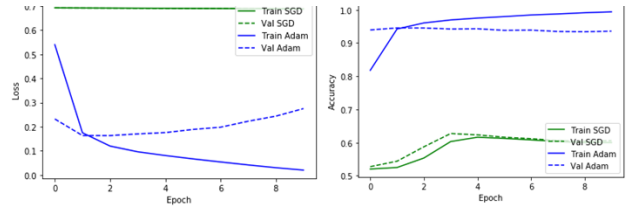


Fig. 7. Comparación del error y la exactitud obtenidos con diferentes optimizadores. A la izquierda la función de error y a la derecha el valor de exactitud. En color verde los valores obtenidos con el optimizador SGD y en azul los obtenidos con Adam.

La anterior figura muestra como la curva correspondiente al error en el conjunto de entrenamiento disminuye hasta valores cercanos a 0, mientras que la curva correspondiente al error obtenido con el conjunto de datos de validación aumenta. Por otra parte, el valor de exactitud llega a valores por encima del 90%, aunque se observa que

el modelo tiende a presentar sobreajuste, también conocido como *overfitting*. Esto se debe a que el algoritmo está memorizando los datos de entrenamiento y no es capaz de generalizar lo aprendido al conjunto de validación. Para contrarrestar el efecto del sobreajuste se aplicó una técnica de normalización, conocida como *dropout* [39].

Una vez aplicada la técnica de dropout a la salida de cada capa de la red, con un valor de hiperparámetro igual a 0.5, se obtiene un valor de exactitud igual a 93.71%, mejorando frente al obtenido anteriormente sin aplicar regularización. La siguiente figura muestra la comparación entre los valores de error y exactitud para los modelos con regularización y sin regularización. En la Figura 8 se observa que el modelo entrenado aplicando dropout (en color azul) disminuye la presencia de sobreajuste, aunque no lo elimina totalmente.

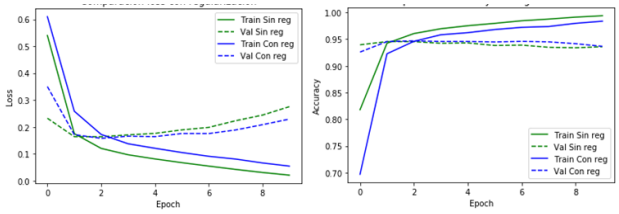


Fig. 8. Comparación del error y la exactitud aplicando regularización.

Otra técnica muy utilizada en la práctica para prevenir sobre ajuste es la de early stopping [40]. Esta técnica permite identificar la iteración o época en que la red comienza a caer en sobreajuste y detiene el entrenamiento en ese momento. La exactitud obtenida después de optimizar el modelo corresponde a un valor de 95%. La Figura 9 presenta la evolución del error en el modelo final de la red neuronal. Se puede observar que el número de épocas de entrenamiento disminuyó a 3 y que el error del conjunto de entrenamiento y del conjunto de pruebas converge en un punto de la gráfica. De manera similar la gráfica de la derecha presenta la exactitud del modelo final, donde se observa que la métrica converge para los dos conjuntos de datos y alcanza un valor de 95%.

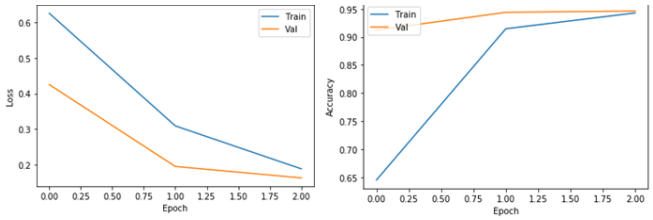


Fig. 9. Error y exactitud en el modelo final de red neuronal.

V. Resultados o Evaluación

El objetivo de las métricas para la evaluación de algoritmos de clasificación es identificar la capacidad de predicción del modelo. Para lograr este objetivo se comparan las clases predichas por el algoritmo para cada ejemplo del conjunto de test con el valor real de la clase y de esta manera se identifica si el modelo logró clasificar correctamente el dato. Una técnica ampliamente usada es la matriz de dispersión, en la que se realiza el conteo de los ejemplos clasificados correcta e incorrectamente, agrupándolos en verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

A. Evaluación de la red neuronal

La evaluación de la red neuronal empleó métricas basadas en la matriz de confusión [41]. La Figura 10 muestra de forma gráfica la matriz de confusión para el modelo final de la red neuronal. Como se puede observar en el gráfico, la red clasificó correctamente el 95% de los casos y solamente falló en el 5%. De este porcentaje de errores la mayor parte corresponde a falsos negativos.

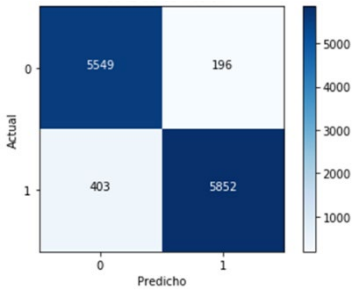


Fig. 10. Matriz de confusión del modelo de red neuronal.

A partir de la matriz de confusión se aplicaron las métricas de evaluación de clasificadores indicadas en la Tabla I.

TABLA I
MÉTRICAS DE EVALUACIÓN DE LA RED NEURONAL

Métrica	Valor
Sensibilidad	97%
Especificidad	93%
Precision	94%
Exhaustividad	97%
Valor-F	95%
Exactitud	**95%**
AUC	98%

En estas métricas se confirma, por una parte, la capacidad de predicción del clasificador obtenida mediante la métrica de exactitud, y por otra parte una tendencia del modelo para predecir con mayor precisión los ejemplos positivos respecto a los negativos. Esto se puede observar teniendo en cuenta que el valor de sensibilidad, que mide la proporción de ejemplos positivos correctamente clasificados, es más alto que el valor de especificidad, que mide la proporción de ejemplos negativos correctamente clasificados.

Otra métrica útil para conocer el desempeño de los modelos de clasificación binarios es el área bajo la curva o *area under the curve* (AUC). Esta medida establece la capacidad del algoritmo para identificar la mayor cantidad de casos verdaderamente positivos sin caer en falsos positivos. En la Figura 11, se muestra la curva ROC obtenida junto con su valor de AUC. El valor de AUC igual a 98% demuestra un equilibrio en el comportamiento de la red neuronal, que se esfuerza en predecir la mayor cantidad de casos verdaderos positivos, evitando caer en falsos positivos.

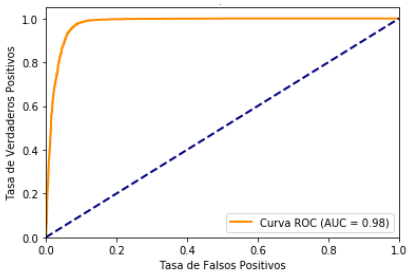


Fig. 11. Curva ROC/AUC. Se presenta la proporción de detección de los verdaderos positivos en el eje y, y la proporción de los falsos positivos en el eje x.

B. Comparación con los modelos de Random Forest y SVM

La Tabla II presenta una comparación de las principales métricas obtenidas luego de aplicar los modelos de redes neuronales, SVM [42] [43] [44] y Random Forest [45] [46]. La tabla muestra que el mejor clasificador es la red neuronal, teniendo en cuenta un valor de exactitud del 95%. Esta es seguida por Random Forest con una exactitud del 92%, y en último lugar se encuentra SVM con un valor del 61%.

TABLA II
MÉTRICAS DE COMPARACIÓN DE LOS MODELOS

Métrica	Red Neuronal	SVM	Random Forest
Sensibilidad	97%	69%	90%
Especificidad	93%	53%	94%
Precisión	94%	60%	93%
Exhaustividad	97%	69%	90%
Valor-F	95%	64%	91%
Exactitud	**95%**	**61%**	**92%**
AUC	98%	61%	92%

La métrica de sensibilidad, que mide la proporción de ejemplos positivos clasificados correctamente muestra un mejor desempeño de la red neuronal frente a Random Forest, y de estos dos frente al SVM. En cuanto a la especificidad, el modelo que clasifica mejor los ejemplos negativos es Random Forest, aunque el valor obtenido por la red neuronal es bastante cercano. Igual sucede con la métrica de precisión que indica la proporción de ejemplos que son verdaderamente positivos. Los valores de exhaustividad o *recall* favorecen por un amplio margen a la red neuronal, que obtiene un valor de 97, frente a un 90 de Random Forest y un 69 de SVM. El valor-F corresponde a un balance entre precisión y recall, y simplifica el rendimiento de un algoritmo de clasificación en una única métrica. La red neuronal obtiene un valor-F de 95, Random Forest de 91 y SVM de 64. Finalmente, el área bajo la curva (AUC) del modelo de redes neuronales supera de nuevo a los otros algoritmos debido a su tendencia a identificar correctamente los verdaderos positivos y evitar los falsos positivos.

El conjunto de métricas aplicadas demuestra que el modelo entrenado con redes neuronales logró un excelente desempeño, superando el umbral propuesto en los objetivos del proyecto. Por otra parte, el modelo entrenado con Random Forest obtiene también buenos resultados, muy cercanos a los de la red neuronal y se contempla como una buena alternativa a este último modelo. El desempeño del modelo de SVM para este caso se considera muy bajo y se descarta su implementación.

VI. Discusión o Análisis de Resultados

Una vez realizado el pronóstico para el conjunto de 39.277.086 personas, el modelo identifico que 3.494.516 personas se encuentran en riesgo de desarrollar enfermedad renal crónica en Colombia. Esta cifra corresponde al 7% del total de la población colombiana, estimada en 49.834.240 para el año 2018 [47]. La Figura 12 muestra la distribución por sexo de las personas a quienes se les pronosticó riesgo de desarrollar ERC. En esta grafica se puede identificar que el 68.08% corresponde a mujeres, y el 31.92% restante corresponde a hombres.

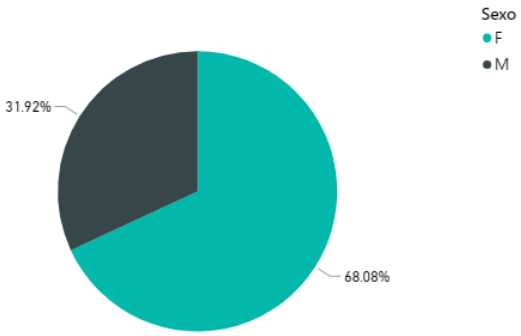


Fig. 12. Distribución de personas en riesgo de desarrollar ERC por sexo.

La Figura 13, presenta la distribución de estas personas por edad. Se puede observar pequeños grupos aislados en edades menores a 25 años, un crecimiento suave entre los 25 y los 45 años y un incremento importante a partir de los 45 años, cuyo pico se centra en los 60 años. A partir de esta edad comienza a disminuir el número de personas.

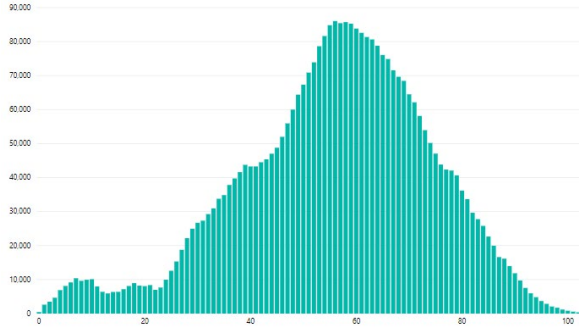


Fig. 13. Distribución de personas en riesgo de desarrollar ERC por edad.

Finalmente, en las Figura 14, se presenta la distribución del porcentaje de personas de acuerdo con su lugar de residencia, en este caso a nivel de departamento. El porcentaje se obtiene tomando como numerador el número de personas identificadas en cada departamento, y como denominador la población total del departamento. Los departamentos que presentan un mayor porcentaje de personas con ERC son Bogotá D.C., Antioquia, Risaralda, Caldas, Santander, Atlántico, Boyacá y Quindío.

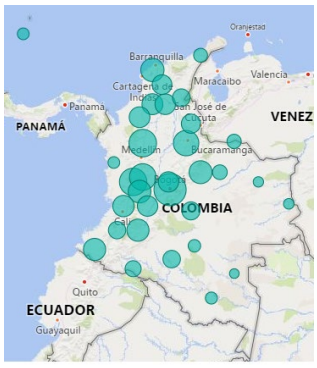


Fig. 14. Mapa de cobertura de la población en riesgo de desarrollar ERC.

VII. Conclusiones

Se comprobó que la red neuronal entrenada con información demográfica y de diagnósticos de enfermedades, es capaz de pronosticar el riesgo de desarrollar enfermedad renal crónica con una exactitud del 95%, demostrando de esta manera la hipótesis planteada inicialmente. Este resultado se obtuvo entrenando una red neuronal con 5 capas: una capa de entrada con 7.492 neuronas, correspondientes a las variables o características del modelo, 3 capas ocultas con 500, 100 y 50 neuronas respectivamente, y una capa de salida con una única neurona que representa la clase del problema de clasificación binario. Para el entrenamiento de la red se empleó una función de activación *ReLU* en las capas ocultas y función sigmoide para hallar la probabilidad en la capa de salida. Como algoritmo de entrenamiento del modelo se utilizó Adam, que demostró una mayor velocidad de convergencia frente a otros algoritmos más tradicionales como *gradient descent* y *stochastic gradient descent* (SGD). Para combatir el efecto del sobreajuste en la red se utilizó regularización haciendo uso de la técnica de *dropout*, en conjunto con *early stopping*.

La evaluación del clasificador se llevó a cabo empleando métricas derivadas de la matriz de confusión, como sensibilidad, especificidad, precisión, exhaustividad y área bajo la curva (AUC). El resultado de sensibilidad, igual a 97%, y de especificidad, igual a 93%, muestran una tendencia de la red para clasificar mejor los ejemplos positivos, frente a los negativos. Por otra parte, el valor de AUC igual a 98% demuestra un equilibrio en el comportamiento de la red neuronal, que se esfuerza en predecir la mayor cantidad de casos verdaderos positivos, evitando caer en falsos positivos.

Además de la red neuronal, se realizó el entrenamiento de otros modelos de aprendizaje automático como Random Forest y máquinas de soporte de vectores (SVM) y se obtuvieron valores de exactitud del 92% y del 61% respectivamente. El modelo de Random Forest obtiene buenos resultados, muy cercanos a los de la red neuronal y se contempla como una alternativa a este último modelo. El desempeño del modelo de SVM para este caso se considera muy bajo y se descarta su implementación para pronosticar nuevos casos de ERC.

El clasificador con redes neuronales demostró su capacidad para aprender a identificar factores de riesgo de la enfermedad [48] y luego aplicar este conocimiento en el pronóstico de nuevos casos. El número de personas a quienes el modelo predice en riesgo de desarrollar enfermedad renal crónica en Colombia es de 3.494.516, es decir un 7% del total de la población. A partir del identificador único de la persona es posible realizar el seguimiento por parte de las entidades aseguradoras con el fin de aplicar actividades preventivas que permitan minimizar el riesgo de desarrollar ERC.

El resultado positivo obtenido en el presente trabajo plantea la posibilidad de utilizar clasificadores con redes neuronales para realizar el pronóstico del riesgo de desarrollar otras enfermedades, especialmente aquellas que causan alta mortalidad entre la población, y aquellas que generan altos costos para el sistema de salud, como cáncer, enfermedades cardiovasculares y diabetes. Es posible ampliar el conjunto de variables agregando otros factores de riesgo como los antecedentes familiares. Los antecedentes familiares han demostrado tener un gran impacto como factor de riesgo, especialmente en enfermedades de tipo cardiovascular [49]. Esta información se podría obtener a partir de las bases de datos de afiliaciones a salud, en donde se indica el parentesco entre el afiliado y sus beneficiarios.

Otros datos de interés para el entrenamiento de los clasificadores son los resultados de las pruebas de laboratorio y los medicamentos suministrados al paciente. Para obtener estas variables es necesario integrar otras bases de datos en las que se encuentra información para grupos de población con patologías específicas. También se puede agregar información relacionada con defunciones para realizar análisis conjuntos de mortalidad y morbilidad. Estos análisis pueden aportar información relevante sobre la relación entre el historial médico del paciente y su causa de muerte.

Los clasificadores entrenados para pronosticar enfermedades específicas se pueden integrar para producir un sistema de medición del riesgo en forma de API o servicio web. A este sistema podrían acceder las entidades prestadoras de servicios de salud para cargar la información de sus pacientes, o incluso los mismos ciudadanos, con el fin de identificar el nivel de riesgo individual para desarrollar este conjunto de enfermedades.

Referencias

[1] Fondo Colombiano de Enfermedades de Alto Costo, «Enfermedad Renal Crónica ERC,» 2014. [En línea]. Available: http://www.cuentadealtocosto.org/index.php/patologias/9-patologias/35-enfermedad-renal-cronica-erc.

[2] Ministerio de Salud y Protección Social, Guía de Práctica Clínica para el diagnóstico y tratamiento de la Enfermedad Renal Crónica, Bogotá, 2016.

[3] Organización Panamericana de la Salud, «La OPS/OMS y la Sociedad Latinoamericana de Nefrología llaman a prevenir la enfermedad renal y a mejorar el acceso al tratamiento,» 2015. [En línea]. Available: https://www.paho.org/hq/index.php?option=com_content&view=article&id=10542:2015-opsoms-sociedad-latinoamericana-nefrologia-enfermedad-renal-mejorar-tratamiento&Itemid=1926.

[4] Fondo Colombiano de Enfermedades de Alto Costo, Situación de la enfermedad renal crónica, la hipertensión arterial y la diabetes mellitus en Colombia, Bogotá, 2017.

[5] W. S. McCulloch y W. Pitts, «A logical calculus of ideas immanent in nervous,» *Bulletin of Mathematical Biophysics,* vol. 5, p. 115–133, 1943.

[6] K. Hornik, M. Stinchcombe y H. White, «Multilayer feedforward networks are universal approximators,» *Neural Networks,* vol. 2, pp. 359-366, 1989.

[7] G. E. Hinton, S. Osindero y Y. Teh, «A fast learning algorithm for deep belief nets,» *Neural Computation,* vol. 18, p. 1527–1554, 2006.

[8] A. Samuel, «Some Studies in Machine Learning Using the Game of Checkers,» *IBM Journal of Research and Developmen,* vol. 3, nº 3, pp. 210-29, 1959.

[9] K. Tretyakov, «Machine learning techniques in spam filtering.» *Data Mining Problem-oriented Seminar, MTAT,* vol. 3, nº 177, pp. 60-79, 2004.

[10] D. West y V. West, «Improving diagnostic accuracy using a hierarchical neural network to model decision subtasks,» *International journal of medical informatics,* vol. 57, nº 1, pp. 41-55, 2000.

[11] Ministerio de Salud y Protección Social, «Sistema de Información de Prestaciones de Salud,» 13 mayo 2019. [En línea]. Available: https://www.minsalud.gov.co/proteccionsocial/Paginas/rips.aspx.

[12] Ministerio de Salud y Protección Social, «Resolución 3374 de 2000,» 13 mayo 2019. [En línea]. Available: https://www.minsalud.gov.co/sites/rid/Lists/BibliotecaDigital/RIDE/DE/DIJ/Resoluci%C3%B3n_3374_de_2000.pdf.

[13] Ministerio de Salud y Protección Social, «Sistema Integrado de Información de la Protección Social,» 28 abril 2019. [En línea]. Available: https://www.sispro.gov.co/Pages/Home.aspx.

[14] Ministerio de Salud y Protección Social, «Misión Institucional,» 13 mayo 2019. [En línea]. Available: https://www.minsalud.gov.co/Ministerio/Institucional/Paginas/mision-vision-principios.aspx.

[15] W. Yu, T. Liu, R. Valdez, M. Gwinn y M. J. Khoury, «Application of support vector Machine modeling for prediction of common diseases: the case of diabetes and pre-diabetes.,» *BMC medical informatics and*

decision making, vol. 10, nº 1, p. 16, 2010.

[16] B. Magnin, L. Mesrob, S. Kinkingnéhun, M. Pélégrini-Issac, O. Colliot, M. Sarazin y H. Benali, «Support vector machine-based classification of Alzheimer's disease from whole-brain anatomical MRI,» *Neuroradiology,* vol. 51, nº 2, pp. 73-83, 2009.

[17] I. S. F. Dessai, «Intelligent heart disease prediction system using probabilistic neural network.,» *International Journal on Advanced Computer Theory and Engineering (IJACTE),* vol. 2, nº 3, pp. 2319-2526, 2013.

[18] Z. H. Zhou, Y. Jiang, Y. B. Yang y S. F. Chen, «Lung cancer cell identification based on artificial neural network ensembles,» *Artificial Intelligence in Medicine,* vol. 24, nº 1, pp. 25-36, 2002.

[19] Y. Cao, Z. D. Hu, X. F. Liu, A. M. Deng y C. J. Hu, «An MLP classifier for prediction of HBV-induced liver cirrhosis using routinely available clinical parameters,» *Disease markers,* vol. 35, nº 6, pp. 653-660, 2013.

[20] K. Kumar y B. Abhishek, «Artificial neural networks for diagnosis of kidney stones disease,» *Information Technology and Computer Science,* vol. 1, pp. 41-48, 2009.

[21] H. Zhang, C. L. Hung, W. C. C. Chu, P. F. Chiu y C. Y. Tang, «Chronic Kidney Disease Survival Prediction with Artificial Neural Networks,» *2018 IEEE International Conference on Bioinformatics and Biomedicine,* pp. 1351-1356, 2018.

[22] E. H. A. Rady y A. S. Anwar, «Prediction of kidney disease stages using data mining algorithms,» *Informatics in Medicine Unlocked,* p. 100178, 2019.

[23] J. Xiao, R. Ding, X. Xu, H. Guan, X. Feng, T. Sun y Z. Ye, «Comparison and development of machine learning tools in the prediction of chronic kidney disease progression,» *Journal of translational medicine,* vol. 17, nº 1, p. 119, 2019.

[24] A. Soldevila, Análisis de la progresión de la enfermedad renal crónica avanzada mediante técnicas de aprendizaje máquina, Valencia, 2017.

[25] S. Hore, S. Chatterjee, R. K. Shaw, N. Dey y J. Virmani, «Detection of chronic kidney disease: A NN-GA-based approach,» *Nature Inspired Computing,* pp. 109-115, 2018.

[26] A. Al Imran, M. N. Amin y F. T. Johora, «Classification of Chronic Kidney Disease using Logistic Regression, Feedforward Neural Network and Wide & Deep Learning,» *2018 International Conference on Innovation in Engineering and Technology,* pp. 1-6, 2018.

[27] H. Kriplani, B. Patel y S. Roy, «Prediction of Chronic Kidney Diseases Using Deep Artificial Neural Network Technique,» *Computer Aided Intervention and Diagnostics in Clinical and Medical Images,* pp. 179-187, 2019.

[28] Y. Ren, H. Fei, X. Liang, D. Ji y M. Cheng, «A hybrid neural network model for predicting kidney disease in hypertension patients based on electronic health records,» *BMC Medical Informatics and Decision Making,* vol. 19, nº 2, p. 51, 2019.

[29] IBM Corporation, Manual CRISP-DM de IBM SPSS, 2012.

[30] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer y R. Wirth, Step-by-step data mining guide, 2000.

[31] M. Nielsen, Neural Networks and Deep Learning, San Francisco, CA, USA: Determination press, 2015.

[32] A. Gulli y S. Pal, Deep Learning with Keras, Packt Publishing Ltd, 2017.

[33] S. Abrahams, D. Hafner, E. Erwitt y A. Scarpinelli, TensorFlow for Machine Intelligence: A Hands-on Introduction to Learning Algorithms, Bleeding Edge Press, 2016.

[34] D. Rumelhart, G. Hinton y R. Williams, «Learning representations by back-propagating errors.,» *Nature,* vol. 323, p. 533–536, 1986.

[35] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, MIT Press, 2016.

[36] K. Jarrett, K. Kavukcuoglu y Y. LeCun, «What is the best multi-stage architecture for object recognition?,» *2009 IEEE 12th international conference on computer vision,* pp. 2146-2153, 2009.

[37] D. P. Kingma y J. Ba, «Adam: A method for stochastic optimization,» *arXiv preprint arXiv:1412.6980,* 2014.

[38] G. Hinton, «Neural networks for machine learning,» *Coursera, video lectures,* 2012.

[39] N. Srivastava, «Improving neural networks with dropout,» *Universidad de Toronto,* 2013.

[40] C. M. Bishop, «Regularization and complexity control in feed-forward networks,» *Proceedings International Conference on Artificial Neural Networks,* p. 141–148, 1995.

[41] T. Fawcett, «An introduction to ROC analysis,» *Pattern Recognition Letters,* vol. 27, nº 8, p. 861–874, 2006.

[42] V. Vapnik, Estimation of Dependences Based on Empirical Data, New York: Springer-Verlag, 1982.

[43] C. Cortes y V. Vapnik, «Support-vector networks,» *Machine learning,* vol. 20, nº 3, pp. 273-297, 1995.

[44] G. James, D. Witten, T. Hastie y R. Tibshirani, An Introduction to Statistical Leaerning with Applications in R, New York: Springer, 2013.

[45] L. Breiman, «Random forests,» *Machine Learning,* vol. 45, pp. 5-32, 2001.

[46] L. Breiman, «Bagging predictors,» *Machine Learning,* vol. 24, nº 2, pp. 123-140, 1994.

[47] DANE, «Proyecciones de población,» 11 mayo 2019. [En línea]. Available: https://www.dane.gov.co/index.php/estadisticas-por-tema/demografia-y-poblacion/proyecciones-de-poblacion.

[48] J. C. Flores, M. Alvo, H. Borja, J. Morales, J. Vega, C. Zúñiga, H. Müller y J. Münzenmayer, «Enfermedad renal crónica: Clasificación, identificación, manejo y complicaciones,» *Revista médica de Chile,* vol. 137, nº 1, pp. 137-177, 2009.

[49] L. Rodríguez, M. E. Díaz, V. Ruiz, H. Hernández, V. Herrera y M. Montero, «Factores de riesgo cardiovascular y su relación con la hipertensión arterial en adolescentes,» *Revista Cubana de Medicina,* vol. 53, nº 1, pp. 25-36, 2014.