Sabrina Lourdes Cabral

Desarrollo de un sistema basado en aprendizaje automático para el sector hortícola

2023

Instituto de Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

Desarrollo de un sistema basado en aprendizaje automático para el sector hortícola.

Sabrina Lourdes Cabral

Florencio Varela, diciembre de 2023

Estudiante

Sabrina Lourdes Cabral

Sabrina.cabral1995@gmail.com

Organización donde se realiza la Práctica Profesional Supervisada

Universidad Nacional Arturo Jauretche.

Av. Calchaquí N. ° 6200, Florencio Varela, (1888) Buenos Aires, Argentina

+54 11 4275 6100

Tutor organizacional

Dr. Ing. Cappelletti, Marcelo

Correo electrónico: mcappelletti@unaj.edu.ar

Docente Supervisor

Ing. Salina, Mauro

Correo electrónico: mdsalina@unaj.edu.ar

Docente tutor del Taller de Apoyo para la Producción de Textos Académicos

Prof. Lavigna Lia

Correo electrónico: llavigna@unaj.edu.ar

Coordinador de la Carrera de Ingeniería en Informática

Dr. Ing. Morales, Martín

Correo electrónico: martin.morales@unaj.edu.ar

Contenido

Índice de Figuras

Resumen

En el presente trabajo se desarrolló un sistema basado en aprendizaje automático que permite la optimización y aprovechamiento del suelo para cultivos hortícolas. Para ello, se diseñaron tres modelos de aprendizaje automático, que en base a diversos parámetros del suelo determinó cuál es el cultivo más apropiado a cosechar. Estos modelos abarcaron la regresión logística, la máquina de soporte vectorial y una red neuronal. Por otro lado, se exploraron diversos métodos para analizar el set de datos empleado. Además, se utilizaron técnicas de validación cruzada para estimar el rendimiento de los modelos y se evaluó el comportamiento del mismo ante datos no conocidos a partir de sus métricas. Por otro lado, se diseñó y desarrolló una aplicación para la plataforma Android con el propósito de que los usuarios puedan visualizar diversos parámetros del suelo de su huerta. Esta aplicación, mediante la integración del modelo basado en una red neuronal, ofrece recomendaciones sobre el cultivo más adecuado al suelo, en función de los parámetros anteriormente mencionados.

Abstract

In this study, a machine learning-based system was developed to optimise and use soil for horticultural crops. To achieve this, three machine learning models were designed, which, based on various soil parameters, determined the most suitable crop for harvesting. These models encompass logistic regression, support vector machines, and a neural network. In addition, various methods were explored to analyse the employed dataset. Cross-validation techniques were employed to estimate the models' performance, and their behaviour was assessed against unknown data using metrics. Furthermore, an application was designed and developed for the Android platform to enable users to visualise various parameters of their garden soil. This application, through the integration of a neural network-based model, provides recommendations for the most suitable crop based on the soil parameters mentioned earlier.

Dedicatorias y agradecimientos

En primer lugar, quiero agradecer a la Universidad Nacional Arturo Jauretche por comenzar mi formación técnica y profesional, me proporcionó saberes y conocimientos que aplico en mi día a día profesional. Además de dotarme de pensamiento crítico y analítico. Por otro lado, quiero agradecer a mis tutores por acompañarme durante este proceso de formación, en especial a mi tutora TAPTA Lavigna Lía, por su dedicación y velocidad con la que corrigió mis informes y por tomarse la molestia de explicar en más de una oportunidad algunas cuestiones que escapaban de mis conocimientos.

También, quisiera agradecer a algunos docentes que marcaron mi paso por la Universidad por su enorme compromiso con el aprendizaje de los estudiantes como Gainle, German; Baum, Lorena; Bracho, Oscar y Sabatino, Daniel.

Finalmente, y no menos importante, quisiera agradecer a mi familia por acompañarme siempre, en especial a mi hermana Belén y a mi mamá Nélida que sin su constante apoyo y ayuda me habría sido más difícil estudiar de lo que fue. Por otro lado, agradecer a los amigos de la facultad con los que compartí estos años de aprendizaje por su ayuda y apoyo en los momentos que me fue necesario, sobre todo a Cristian y a Mauro.

1. Introducción

La presente Práctica Profesional Supervisada (PPS) se desarrolló en el marco del Proyecto de Vinculación de la Universidad Nacional Arturo Jauretche UNAJ VINCULA 2022 (Resolución Rectoral N. ° 300/22), cuyo título es “Desarrollo de soluciones tecnológicas de bajo costo a problemáticas en el sector hortícola de la zona de influencia de la UNAJ”, bajo la Dirección del Dr. Ing. Marcelo Cappelletti.

El principal enfoque de la práctica se basó en el desarrollo de una aplicación para dispositivos móviles, con sistemas operativos Android, versión 7 y versiones superiores, que recomiende en base a determinadas características del suelo, cuál es el cultivo óptimo para desarrollarse en dicho suelo.

Para poder llevar a cabo la tarea de clasificación, la aplicación se integró a un modelo de aprendizaje, que tiene la capacidad de identificar patrones en grandes conjuntos de datos. Este modelo puede ayudar a los productores a advertir sobre el comportamiento actual y futuro de un determinado cultivo, y en base a ello tomar decisiones apropiadas con el propósito de obtener cultivos sanos y buenos rendimientos. Además, el modelo puede recomendar qué cultivo es el más adecuado, para crecer en una huerta en función de los parámetros del suelo tales como humedad, temperatura, pH, fósforo, nitrógeno, potasio, entre otros.

1.1. Motivación

La Universidad Nacional Arturo Jauretche (UNAJ) se localiza en el partido de Florencio Varela, uno de los municipios con mayor superficie rural dentro del Conurbano Bonaerense. En la actualidad, esta región, cuenta con aproximadamente unos 500 pequeños, medianos y grandes productores hortícolas, florícolas y frutícolas. Es importante aclarar, que la producción frutihortícola del país se concentra mayormente en el cinturón verde de Buenos Aires, definido como una zona periurbana de producción, motivada fundamentalmente por la reducción de los costos de traslado y por la perecibilidad de ciertos productos de hoja. En general, estas actividades están basadas fundamentalmente en los cultivos intensivos, donde se busca maximizar la producción en espacios reducidos utilizando un solo tipo de producto a la vez. En particular, dentro de la producción hortícola, las modalidades de producción son tanto a campo abierto como bajo cubierta (invernadero). Una de las particularidades sobresalientes de esta zona rural es que limita con áreas de similares características con los partidos vecinos, lo que transforma a la subregión en estratégica para

la producción de alimentos frescos de origen vegetal, tanto para el consumo directo como su industrialización.

En los últimos años, la solidez del sistema productivo de los pequeños y medianos productores agroindustriales de la región, se ha visto resentida por los vaivenes de la economía y en algunos casos por condiciones climáticas adversas que afectaron en mayor proporción a los cultivos realizados a la intemperie, con el consiguiente aumento de los precios como consecuencia de la disminución de la oferta.

Se pueden identificar una serie de problemas que presentan en general los productores de la región, los dos más importantes son los siguientes:

- Falta de información precisa, actual e histórica de variables de interés, debido a que esta actividad requiere, entre otras cosas, de la disponibilidad oportuna de datos, tanto del clima, como del suelo y del agua.
- Insuficiente innovación tecnológica, dado que existe una carencia importante en cuanto a la implementación de la tecnología informática para los cultivos.

1.2. Objetivos

El objetivo principal que se propone en el Plan de Trabajo consiste en el estudio diseño y desarrollo de un sistema basado en algoritmos de aprendizaje automático, que es implementado en una aplicación móvil y que permite llevar a cabo acciones de control, monitoreo y gestión de magnitudes climatológicas, del estado del suelo y del agua. El sistema fue diseñado de forma flexible para poder adaptarlo a otro tipo de aplicaciones.

1.2.1. Objetivos secundarios

Como objetivo secundario se formaron recursos humanos en el campo de la Inteligencia Artificial, lo cual es de una relevancia evidente para el desarrollo tecnológico del territorio de la UNAJ.

1.2.2. Objetivos específicos

La premisa del presente trabajo, sostiene que, mediante una mayor investigación en la articulación de la inteligencia artificial con el desarrollo de aplicaciones móviles, se pueden generar nuevas y mejores propuestas tecnológicas, que contribuyan activamente al desarrollo del sector hortícola, especialmente a los pequeños productores, ya que son soluciones tecnológicas de bajo costo que no requieren de herramientas o dispositivos sofisticados.

1.2.3. Tareas llevadas a cabo

Teniendo en cuenta, la premisa planteada, a continuación, se describen cuáles fueron las tareas desarrolladas:

- **Investigación sobre Inteligencia Artificial, ramas de la misma y análisis de las herramientas de aprendizaje:** durante el desarrollo de esta tarea se abordó el concepto de inteligencia artificial y cuáles son las ramas que posee. Además, se investigó y se analizó información relacionada con las herramientas de aprendizaje automático, con el propósito de adquirir conocimientos y capacidades específicas sobre los últimos avances referidos a esta temática (fundamentos, evolución, características, ventajas y desventajas, aplicaciones, estructura, eficiencias, etc.).
- **Análisis y pre-procesamiento del conjunto de datos. Investigación de los diversos modelos de aprendizaje:** existen diversas formas de llevar a cabo este procesamiento o "limpieza" de datos, algunas de ellas son mediante la reducción de la dimensión, la normalización, la detección de valores atípicos, un análisis estadístico, gráfico de los datos, el análisis de datos faltantes, entre otras operaciones. El principal objetivo de esta tarea, fue determinar, si en el conjunto de datos, existen datos incompletos, atípicos o si las clases se encuentran desbalanceadas, con el fin de garantizar que el conjunto de datos se encuentre "limpio" y que, además, proporcione la suficiente información significativa y relevante para ser empleada.

  Por otro lado, también, se analizó y comparó los diferentes algoritmos de aprendizaje automático, en particular, los algoritmos de regresión y clasificación, con el fin de determinar cuál es el más adecuado. Se estudió las librerías scikit-learn y Tensor-Flow para la creación de diversos modelos y se analizó el desempeño de cada modelo creado.
- **Desarrollo de interfaz de usuario:** en esta etapa se llevó a cabo una investigación de las diversas formas de integración entre una aplicación Android y el modelo desarrollado para la clasificación de cultivos. Posteriormente, se llevó a cabo el desarrollo de dicha aplicación que, en base a los datos de pH, humedad, fósforo, nitrógeno, potasio y lluvia, puede determinar qué tipo de cultivo es apropiado para las características del suelo. Además, también permite la visualización de los datos de la huerta en tiempo real.
- **Testear y analizar el funcionamiento de la aplicación.** Para determinar el correcto funcionamiento de la aplicación, se empleó la herramienta de Firebase, como modelo de base de datos Cloud. En ella se cargaron los datos pertenecientes a huertas, que

no formaron parte de los datos de entrenamiento y, posteriormente, la aplicación consulta dichos datos y, mediante el modelo construido se lleva a cabo la clasificación, para que finalmente se muestre al usuario el cultivo propicio para dicho suelo, además de algunos datos relacionados a él.

2. Marco teórico

2.1. Inteligencia artificial IA

La inteligencia artificial busca imitar el razonamiento humano mediante la creación de algoritmos y sistemas que tengan la capacidad de ejecutar, desde tareas simples a complejas, sin la intervención de un humano. Pero, no todas las tareas que no requieren la intervención humana implican la utilización de una IA, ya que algunas de ellas podrían ser resueltas por herramientas de automatización de procesos, como lo son los BPMN. La IA tiene la particularidad, que durante el proceso que realiza su tarea, tiene la capacidad de aprender, razonar, reconocer patrones y finalmente tomar decisiones para la resolución del problema. Su principal objetivo es el desarrollo de modelos que, mediante el procesamiento de información, estos puedan aprender y generar la capacidad de tomar decisiones sin la intervención directa de los humanos. Es importante aclarar, que mientras más información procese mejor será el rendimiento del modelo.

“Actualmente, la IA abarca una gran variedad de subcampos (ver figura 1), que van desde lo general (aprendizaje y percepción) hasta lo específico, como jugar al ajedrez, probar teoremas matemáticos, escribir poesía, conducir un automóvil en una calle concurrida y diagnosticar enfermedades. La IA es relevante para cualquier tarea intelectual; es verdaderamente un campo universal.” (Russell and Norvig, 2009)

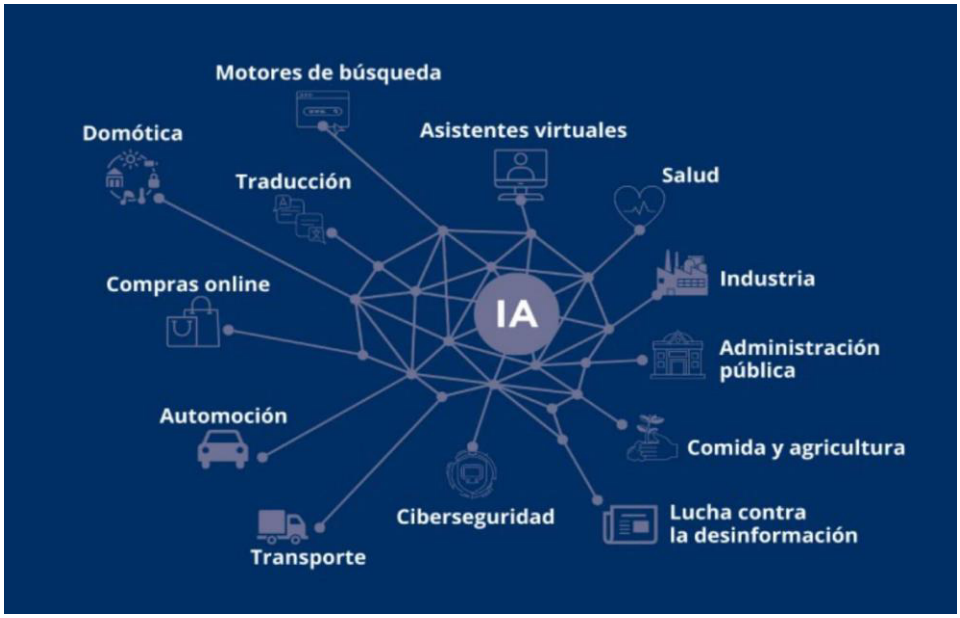


Figura 1. Campos de aplicación de la Inteligencia Artificial.
Fuente: Recuperado de https://planderecuperacion.gob.es/noticias/que-es-inteligencia-artificial-ia-prtr

2.2. Machine Learning.

Machine Learning (ML) es una rama de la inteligencia artificial que tiene como objetivo el desarrollo de modelos capaces de detectar automáticamente patrones en los datos y realizar predicciones, especialmente, en contextos de incertidumbre. Estas predicciones se aplican generalmente a un futuro cercano, siempre que el contexto en el cual se obtuvieron los datos, no haya cambiado bruscamente. Además, también puede ser de utilidad en la toma de decisiones en situaciones de incertidumbre. Es importante resaltar que el aprendizaje automático es un cambio de paradigma con respecto a la programación tradicional. En esta última, las instrucciones se dan de forma explícita a la computadora, mientras que, en el aprendizaje automático, los algoritmos no se programan de manera explícita, en cambio, estos algoritmos aprenden de los datos (Ver figura 2) y ajustan automáticamente sus parámetros para optimizar sus resultados. Cuando se trabaja con conjuntos de datos grandes, se logra mejorar la precisión y, de esta manera, obtener predicciones más acertadas.

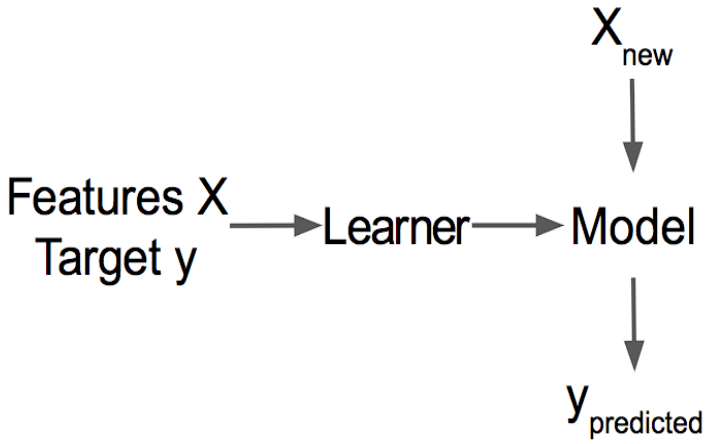


Figura 2. Aprendizaje de un modelo a partir de datos.
Fuente: Molnar, C. (2023) "Introduction" en *Interpretable Machine Learning*. Capítulo 2.3. p 16.

Asimismo, el aprendizaje automático está estrechamente relacionado con la teoría de la probabilidad, especialmente, en aquellos modelos que emplean algoritmos que hacen uso explícito de esta herramienta para abordar problemas que involucran incertidumbre, un ejemplo de ello son los clasificadores bayesianos. En estos casos, la probabilidad permite a los modelos cuantificar y modelar dicha incertidumbre.

Sin embargo, es importante resaltar que, si bien algunos modelos de aprendizaje automático emplean un enfoque probabilístico para llevar a cabo predicciones, existen otros algoritmos que no hacen uso explícito de la probabilidad, como lo son máquinas de soporte vectorial o los árboles de decisión. Para aquellos casos en que las ML, si se emplean los teoremas de probabilidad como herramienta, es fundamental comprender la diferencia entre ambos enfoques, ya que no solo difiere en terminología, sino también en cuanto a finalidad. Las ML se centran en realizar predicciones basadas en grandes conjuntos de datos, mientras que el modelo estadístico busca reconocer y caracterizar las relaciones entre datos y las variables resultado, haciendo énfasis en la importancia de dichas relaciones, además de realizar predicciones.

2.3. Tipos de aprendizaje

El aprendizaje automático se segmenta generalmente en dos áreas principales, el aprendizaje supervisado, que posee un enfoque predictivo o supervisado y, el segundo tipo de aprendizaje, aborda un enfoque descriptivo o no supervisado.

2.3.1. Aprendizaje Supervisado

En este tipo de aprendizaje, se le indica al algoritmo qué es lo que debe predecir, por lo tanto, se trabaja con conjuntos de datos etiquetados y que ya tienen definido un resultado;

es decir, que para cada entrada X se tiene una salida Y definida, la cual se encuentra representada por la siguiente fórmula:

$$D = \{(xi, yi)\} \frac{N}{i = 1}$$

Donde D indica al conjunto de datos de entrenamiento y N es el número de iteraciones en el entrenamiento del modelo. En cada iteración, el modelo evalúa su desempeño en la tarea que está aprendiendo y ajusta los sesgos y pesos (parámetros) gradualmente, con el objetivo que las predicciones sean lo más cercanas posible a las respuestas correctas.

Cabe aclarar, que cada conjunto de datos es altamente variable y depende del caso de estudio. Se pueden trabajar con dataset que están conformados por vectores de n características o covariables (como es el caso de estudio del presente trabajo), así como también, con datos de estructuras más complejas, como imágenes, series temporales, entre otros.

2.3.2. Aprendizaje no supervisado

En este caso, se trabaja con un conjunto de datos que no se encuentra etiquetado, por lo tanto, no se sabe el resultado esperado. Consecuentemente, el modelo debe reconocer ciertos patrones o estructuras ocultas en los datos, que ocurren con mayor frecuencia.

Este tipo de aprendizaje es sumamente útil, cuando se trabaja con un volumen de datos grande que no se encuentra etiquetado y no se puede determinar con facilidad la relación existente entre los datos. Posee muchas aplicaciones, una de ellas es la técnica de clustering, que se basa en agrupar aquellos elementos que poseen características similares en grupos o clústeres, esto permite una mejor compresión en la distribución y estructura de los datos. Otra técnica, se basa en reducir el conjunto de datos, para que solo prevalezcan aquellos datos que pueden ser considerados de interés. También, se lo puede utilizar para buscar valores estadísticos que describan a los datos, o se lo emplea para detectar valores atípicos o anomalías. Este último caso de uso, normalmente es utilizado para identificar fraudes en transacciones financieras.

2.4. Algoritmos aplicados en los modelos de aprendizaje automático.

Un algoritmo de aprendizaje automático es un programa diseñado para entrenar un modelo a partir de los datos de entrada proporcionados. Es relevante señalar, que existen diferentes tipos de clasificación:

- Clasificación binaria: es aplicado en problemas en los que el resultado puede ser uno de dos valores, los cuales son mutuamente excluyentes. Un ejemplo de uso es para determinar si un correo es spam o no, donde el modelo proporcionará los resultados "Sí" para correos que sean spam y "No" para aquellos que no lo son.
- Clasificación multiclase: Cuando se quiere relacionar un conjunto de datos X con una variable dependiente D, se está abordando un problema de clasificación multiclase. En este tipo de clasificación, existen múltiples variables dependientes o categorías y cada conjunto de datos X solo puede pertenecer a una única categoría. Este es el caso de estudio del trabajo.
- Clasificación multietiqueta: Para este caso, cada entrada de datos puede pertenecer a más de una categoría o etiqueta, es decir, que la asociación entre la entrada y la salida, no es un uno a uno, como en la clasificación binaria y multiclase, sino que en este caso la asociación es 1 a n etiquetas. Un ejemplo de uso es el reconocimiento de imágenes, donde una imagen puede contener múltiples elementos que la conforman y se debe clasificar cada uno de ellos. La entrada es una imagen, pero la salida dependerá de la cantidad de objetos diferentes, que estén contenidos en ella.

Existen diferentes tipos de algoritmos de clasificación, y al momento de decidir cuál emplear se debe contemplar la problemática que se está por abordar. En esta sección, se tratarán únicamente aquellos algoritmos que se emplean para problemas de clasificación.

2.4.1. Regresión Logística Multiclase

"La regresión logística es una extensión del modelo de regresión lineal, pero para problemas de clasificación." (Molnar, 2022). La regresión lineal se emplea para predecir valores numéricos continuos, mientras que la regresión logística se la utiliza para predecir si un conjunto de datos pertenece a una clase específica o no.

En la regresión logística se emplea la función Sigmoide (también conocida como función logística, ver figura 3). Dicha función está definida como:

$$logistic(\eta) = \frac{1}{1 + exp\,(-\eta)}$$

Esta función se aplica a la combinación lineal de las variables predictoras, que son los valores de cada una de las características que corresponden a cada categoría y sus pesos. Estos son los coeficientes que indican cuánto influye cada variable en la predicción de pertenecer a la categoría, es decir, en el resultado final. La combinación lineal se la define como:

$$z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

Donde β_0 es el intercepto o sesgo y no se lo multiplica con ninguna variable. Las β_n representan los pesos o coeficientes de las características aprendidas de cada una de las variables y los X_n representan las variables.

La función sigmoide, es utilizada para transformar z en una probabilidad en el rango de 0 a 1. Cuando z toma valores grandes y positivos, la función sigmoide tiende a 1, lo que indica una alta probabilidad de pertenencia a esa clase. Por otro lado, cuando z toma valores grandes y negativos, la función sigmoide se acerca a 0, lo que indica una baja probabilidad de pertenencia a dicha clase. Los valores de salida de esta función oscilan en el rango de [0,1] y, para obtener predicciones discretas, se emplea un umbral, que es de 0,5. Cuando la predicción es superior a 0,5 se la considera como positiva, es decir 1, mientras que cuando es menor, se la considera como negativa, es decir 0.

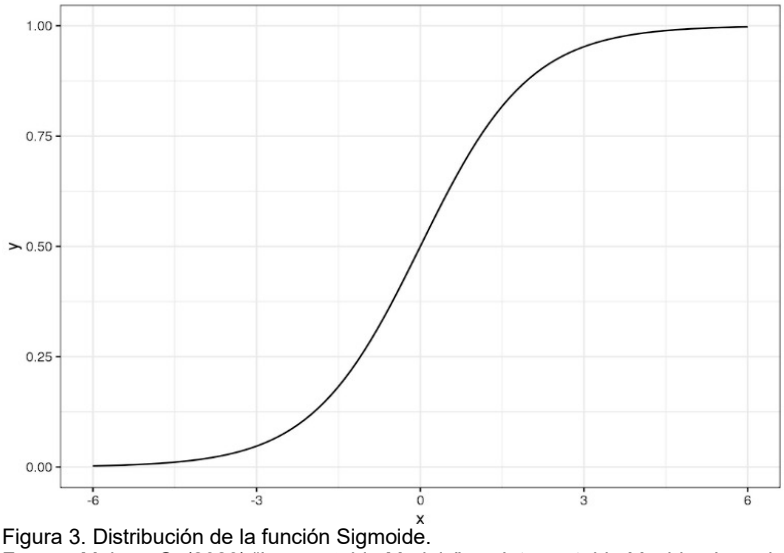


Figura 3. Distribución de la función Sigmoide.
Fuente: Molnar, C. (2023) "Interpretable Models" en *Interpretable Machine Learning*. Capítulo 5.2.1, p 56.

La regresión logística es nativamente binaria, pero puede ser utilizada para la clasificación multiclase, que es el caso de interés. Para lograr esto, se emplea como función de activación la función Softmax, que está definida como:

$$Softamx(a_i) = \frac{e^{ai}}{\sum_{j=1}^{m} e^{a_j}} \ , i = 1, \dots m$$

Donde $Softamx(a_i)$ es la probabilidad de que la instancia pertenezca a dicha clase, ai es la combinación lineal de las variables predictoras x, y los pesos asociados a dicha clase. Por otro lado, se emplea la función exponencial para garantizar que los valores sean positivos,

y $\sum_{j=1}^{m} e^{a_j}$, representa la suma de las exponenciales de todas las combinaciones lineales para todas las clases.

Esta función toma el vector de entrada y normaliza la distribución de probabilidades, que está conformado por las m categorías y la probabilidad de pertenecer a cada una de ellas. De esta forma garantiza que la sumatoria del vector de salida sea igual a 1. Luego, a partir de la función Softmax se puede establecer qué categoría posee el valor más alto y, finalmente, determinar para cada elemento del conjunto de datos de entrada su clase correspondiente.

El algoritmo de regresión logística emplea una función de costo, que la utiliza para cuantificar la diferencia entre las predicciones del modelo respecto a los resultados reales obtenidos de los datos de entrenamiento. El objetivo es minimizar la función de costo, que implica ajustar los pesos del modelo para que las predicciones sean lo más cercanas a las categorías reales.

Durante el proceso de aprendizaje, el algoritmo ajusta los coeficientes en cada iteración con el propósito de minimizar la función de costo. Para ello, puede emplear los siguientes métodos de optimización:

- Descenso de gradiente por lotes: consiste en calcular la gradiente de la función de costo en relación con cada uno de los parámetros (pesos) de los datos de entrenamiento, la definición de la misma es la siguiente:

$$i = i - el\ \nabla J(i)$$

Donde i es el vector de parámetros (pesos) del modelo que se está ajustando, el es la tasa de aprendizaje y $\nabla j_i(i)$ es el gradiente de la función de costo con respecto a los parámetros (i). Este es un vector que contiene las derivadas parciales de J con respecto a cada uno de los parámetros.

Esta técnica solo es empleada cuando el conjunto de datos de entrenamiento es pequeño, ya que, calcula los gradientes para todo el conjunto de datos antes de realizar una actualización de los pesos en el modelo. Esto implica que, para el cálculo del gradiente, se considere todo el conjunto de datos de entrenamiento en cada iteración. Posteriormente, se ajusta gradualmente los pesos i restando el producto entre el gradiente y la tasa de aprendizaje el. Este ajuste, se realiza para cada parámetro del modelo, lo que implica que la función de costos se minimiza. Este proceso puede ser computacionalmente costoso, cuando se trabaja con conjuntos de datos grandes, debido a que deben ser almacenados en memoria.

- Descenso de gradiente Estocástico: consiste en calcular el gradiente de la función de costos de una sola muestra a la vez del conjunto de datos en cada iteración. Esto implica, que la actualización de pesos se haga con mayor frecuencia. La definición de la misma es la siguiente

$$i = i - el\, \nabla J(i;\, X^{(i)}, Y^{(i)})$$

Donde i es el vector de pesos del modelo, el es la tasa de aprendizaje y $\nabla J(i;\, X^{(i)}, Y^{(i)}$ es la gradiente de la función de costos con respecto a los pesos i, empleado las características X y las etiquetas Y.

La frecuencia de actualización en los pesos, hace que la función de costos converja más rápido. “Como gradiente se calcula en base a un único ejemplo de entrenamiento, la superficie de error es más ruidosa. Sin embargo, tiene la ventaja de que puede escapar de los mínimos locales poco profundos más fácilmente” (Rashka and Mirjalilli, 2017).

Esta técnica, es ampliamente usada cuando se trabaja con un conjunto de datos grandes

- Descenso de gradiente de mini lotes: es una técnica que combina el enfoque del Descenso de Gradiente por Lotes y el Descenso de Gradiente Estocástico. Se aplican los cálculos de gradiente a un subconjunto más pequeño de los datos de entrenamiento (el tamaño de los mismos puede variar según el tamaño del conjunto de datos y la memoria disponible). Durante el entrenamiento, el algoritmo calcula el gradiente de la función de costo para cada subconjunto y realiza ajustes en los pesos del modelo de acuerdo con el gradiente calculado. Como aumenta la frecuencia de las actualizaciones de los pesos, se logra una convergencia más rápida.

La elección del tamaño del subconjunto es un hiperparámetro importante y puede afectar el rendimiento y la eficiencia del entrenamiento. Debido a que, un subconjunto pequeño puede proporcionar estimaciones de gradiente más ruidosas, ya que se cuenta con una menor cantidad de ejemplos empleados. Sin embargo, permite una actualización de pesos más frecuente, lo que puede ayudar a acelerar la convergencia del algoritmo. Por otro lado, un subconjunto grande puede proporcionar estimaciones de gradiente más precisas, al utilizar una mayor cantidad de ejemplos. Sin embargo, debido a que el tamaño es grande, las actualizaciones de los pesos son menos frecuentes, lo que puede ralentizar el proceso de convergencia del algoritmo.

- Una alternativa a las técnicas anteriormente mencionadas, es el método de Newton-Raphson, que es un algoritmo de optimización ampliamente conocido y uno de los más antiguos. Aunque puede ser aplicado a una gran variedad de problemas actuales, es importante destacar que su uso puede variar según el contexto. Al igual que los optimizadores anteriores, el objetivo, en este caso, es minimizar la función de costo. La actualización de los parámetros en el método de Newton-Raphson se define mediante la siguiente fórmula:

$$\omega_{n+1} = \omega_n - H^{-1}(\omega_n)\nabla f(\omega_n)$$

Donde ω_{n+1} representa los nuevos pesos calculados en la iteración $n+1$, ω_n son lo coeficientes actuales en la iteración n, H^{-1} es la inversa de la matriz Hessiana, que contiene las segundas derivadas parciales de la función de costos con respecto a los pesos. Es importante destacar que, si H no es positiva puede generar problemas de convergencia. $\nabla f(\omega_n)$ es la gradiente de la función de costos, que es un vector que contiene las primeras derivadas parciales, con respecto a los pesos.

En este método, la matriz Hessiana proporciona información de cómo cambian las derivadas parciales de primer orden de la función de costos con respecto a todas las posibles combinaciones de las características y los pesos del modelo. Esto sirve para entender cómo varía la pendiente de la función de costos, en relación con las actualizaciones en los pesos de las características del modelo. Además, con la matriz Hessiana, se puede determinar cuál es el sentido de la curva de la función. Si todas las segundas derivadas son positivas la región es convexa, lo que indica que se está cerca de un mínimo. Con esta información, el algoritmo ajusta la dirección y el tamaño de los pesos de las características. Este es un proceso iterativo, hasta que se logre cumplir con un criterio de convergencia.

Si bien este método puede lograr una convergencia más rápida que, con otros métodos, es importante aclarar que es computacionalmente costoso, calcular el gradiente y la inversa de la matriz Hessiana, sobre todo si el conjunto de datos posee muchas características.

2.4.2. Support Vector Machine (SVM)

"Es un algoritmo de clasificación de datos lineales y no lineales, que puede emplear un mapeo lineal, si los datos son linealmente separables, o en caso contrario emplea un mapeo no lineal para transformar los datos de entrenamiento originales en una dimensión superior. Dentro de esta nueva dimensión, se busca el hiperplano separador óptimo lineal,

que es un "límite de decisión" que separa las tuplas de una clase de otra. Con un mapeo no lineal apropiado a una dimensión suficientemente alta, los datos de dos clases siempre pueden estar separados por un hiperplano. El SVM encuentra este hiperplano usando vectores de soporte (tuplas de entrenamiento "esenciales") y márgenes (definidos por los vectores de soporte)" (Harrigton, 2012).

El algoritmo SVM, cuando trabaja con un conjunto de datos que posee solo dos categorías, busca definir un hiperplano y, en el caso de que se trabaje con n>2 categorías busca definir los hiperplanos, que tengan un error mínimo de clasificación. Estos hiperplanos son un límite de decisión entre las clases, y son independientes a la cantidad de atributos del conjunto de datos.

En la clasificación binaria, todo lo que se encuentra en un lado del hiperplano pertenece a una clase, mientras que lo que se encuentra en el otro lado, pertenece a la otra clase. Los puntos más cercanos al hiperplano de separación se los denomina vectores de soporte y, con ellos se busca maximizar la distancia desde el hiperplano a los vectores de soporte; a esta distancia se la denomina margen y el hiperplano con mayor margen de separación entre clases, se lo llama hiperplano marginal máximo (MMH). Los vectores son fundamentales o esenciales, se encuentran más cerca del límite decisión y tienen una influencia directa en el cálculo del vector de pesos y en el sesgo, que determinan la posición del hiperplano de separación, ya que el algoritmo trabaja para maximizar el margen (la distancia) entre el hiperplano y estos vectores de soporte.

El clasificador emplea la función de decisión, $f(w^t x + b)$, para definir las líneas o hiperplanos de separación (dependiendo la cantidad de clases que se esté manejando). En la clasificación binaria, esta función, permite describir qué tan cerca se encuentra un punto del hiperplano de separación. Además, de indicar a qué clase pertenece un punto en función del hiperplano, ya que se conoce de qué lado se encuentra con respecto a este. Se puede describir la función de la siguiente forma:

w es el vector de pesos que define la orientación y la dirección de la línea de separación. x es el vector de características del punto que se desea clasificar y b es el término de sesgo (bias), que ajusta la posición de la línea de separación.

La interpretación del resultado de la función es la siguiente:

- $f(w^t x + b) < 0$ el punto x se encuentra en el lado negativo de la línea de separación y ese dato se lo clasifica como negativo, es decir, no pertenece a la clase.

- $f(w^t x + b) > 0$ el punto x se encuentra en el lado positivo de la línea de separación y ese dato se lo clasifica como positivo, es decir, pertenece a la clase.

El proceso de entrenamiento SVM busca encontrar el vector de pesos w y el sesgo b que maximizan el margen, es decir, maximizan la distancia entre el hiperplano de separación y los vectores de soporte. Este proceso se realiza de manera que se minimice el error de clasificación en el conjunto de entrenamiento. En otras palabras, se busca el equilibrio entre lograr un margen amplio para generalizar bien a datos no vistos y minimizar la clasificación incorrecta en los datos de entrenamiento. El SVM ajusta los parámetros w y b de forma iterativa con el objetivo de lograr esta optimización, garantizando así que el hiperplano resultante sea óptimo, tanto en términos de separación como de generalización.

Es importante recalcar, que la complejidad del clasificador se encuentra determinada por el número de vectores de soporte más que por la dimensionalidad de los datos, por lo tanto, las SVM tienden a ser menos propensas al sobreajuste.

Hasta el momento, se ha abordado la explicación de SVM como un clasificador binario que se aplica a datos linealmente separables (ver figura 4). Sin embargo, es necesario ahondar en el concepto de Máquina de Vectores de Soporte (SVM) en un sentido más general, capaz de manejar múltiples clases. Es fundamental destacar, que este algoritmo es inherentemente binario, pero también es aplicable a casos en los que se necesite clasificar en más de una categoría. Además, el SVM es capaz de resolver situaciones en las que los datos no pueden ser separados linealmente (ver figura 5).

En primer lugar, se aborda el último caso mencionado, que es cuando los datos no son linealmente separables. En este tipo de casos, no se pueden hallar los hiperplanos en el espacio de características original, que logre separar las categorías, el SVM emplea una estrategia para transformar estos datos a un espacio de mayor dimensión donde podrían volverse linealmente separables. En este paso se pueden utilizar diversos mapeos no lineales, que buscan transformar la complejidad de las relaciones entre características, a relaciones más distinguibles. Una vez que los datos se han transformado en un nuevo espacio superior, se buscan los hiperplanos de separación en el nuevo espacio. Estos corresponden a superficies de separación no lineales en el espacio original. (Ver figura 6).

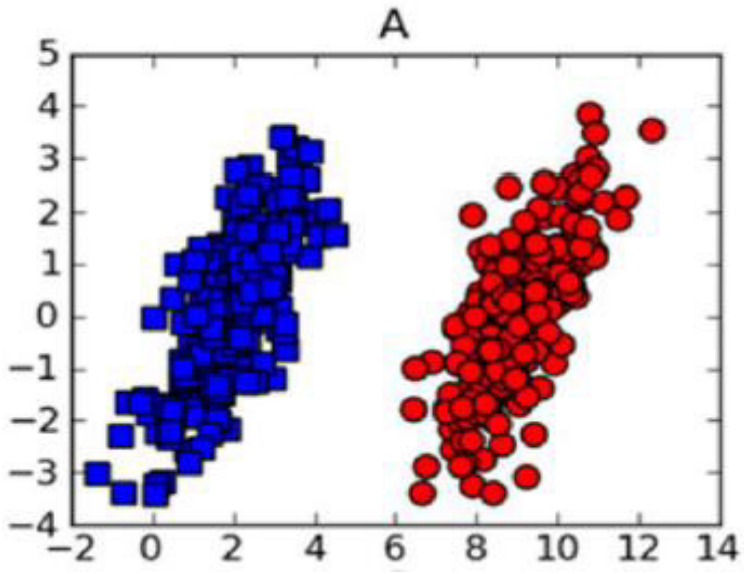


Figura 4. Datos linealmente separables.
Fuente: Harrington, P. (2012) “Support Vector Machine” en *Machine Learning in Action*. Capítulo 6.1, p 103.

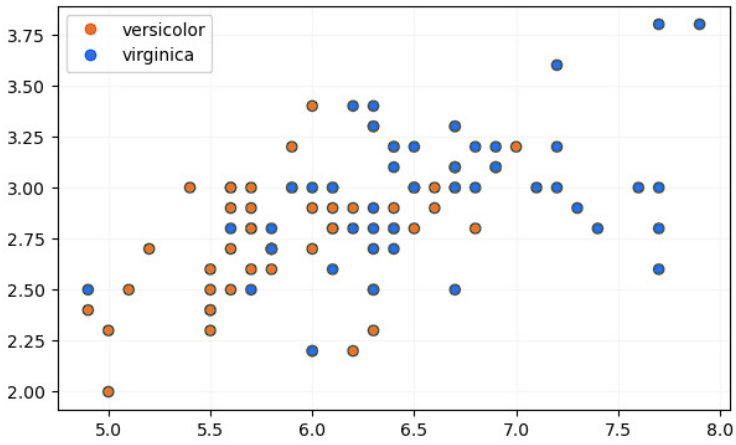


Figura 5. Datos no linealmente separables.
Fuente: Extraído de https://interactivechaos.com/es/manual/tutorial-de-machine-learning/datos-linealmente-no-separables

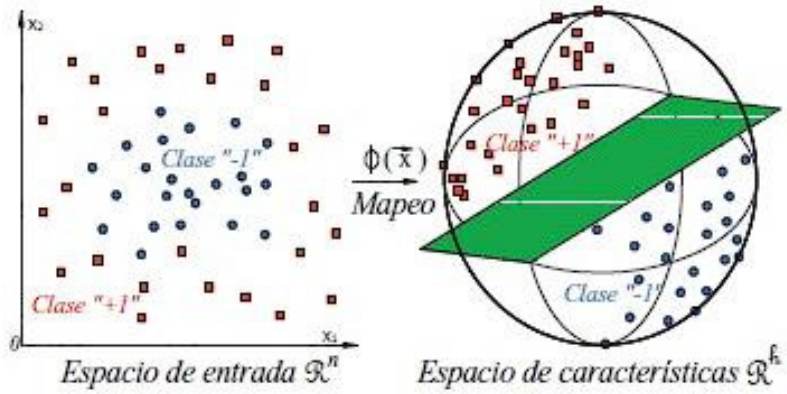


Figura 6. Aplicación de una dimensión superior para la clasificación de los datos.
Fuente: Extraído de http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992019000300059

Mediante la técnica de buscar un espacio de mayor dimensión, donde las clases pueden volverse linealmente separables, se pretende determinar a qué categoría pertenecen los datos de entrada. Sin embargo, este enfoque requiere calcular el producto punto entre las tuplas de entrenamiento y cada vector de soporte, y repetir este proceso hasta encontrar los Hiperplanos Marginales Máximos (MMH). Dicho cálculo puede resultar computacionalmente costoso, especialmente al trabajar con un gran número de vectores de soporte.

Para abordar esta problemática, los SVM emplean funciones kernel. Estas funciones podrían considerarse como una interfaz, que traduce los datos desde un formato complejo a un formato más manejable. En lugar de calcular los productos puntos entre los datos de entrenamiento y los vectores de soporte, se los reemplaza por la evaluación de la función kernel. Esta sustitución permite resolver el problema de manera eficiente en un espacio de alta dimensión, lo que es equivalente a abordar un problema no lineal en un espacio de baja dimensión.

Las funciones de kernel, que sirven para encontrar hiperplanos de separación en un espacio de mayor dimensión, son las siguientes:

- Kernel Polinómico de grado, cuya fórmula es $K(X_i, X_j) = (X_i * X_j + 1)^k$, (X_i, X_j) son vectores de características en la instancia i-ésima y j-ésima en el espacio original y k es el grado del polinomio que determina la complejidad de la transformación. Un valor alto de k significa una transformación a un espacio de mayor dimensión.
  El valor resultante $K(X_i, X_j)$ representa la similitud transformada entre las dos instancias en el espacio de mayor dimensión. Este valor refleja cómo las características de las dos instancias se relacionan en el contexto de una función polinómica de grado k.
- Función de base radial (versión gaussiana): esta toma un vector como entrada y produce un valor numérico (un escalar) como salida. La base radial (versión gaussiana) calcula este valor en función de la distancia del vector a otro punto de referencia. Puede ser la distancia desde el punto (0,0) o la distancia a otro vector en el conjunto de datos. La fórmula es la siguiente:

$$K(X_i, X_j) = \frac{e^{|x_i - x_j|}}{2\sigma^2}$$

  (X_i, X_j) son dos vectores de características en la instancia i-ésima y j-ésima, o es un parámetro que determina la rapidez con la que disminuye la función a cero. Este parámetro afecta el alcance de la función y cuánto influyen los puntos distantes en la similitud calculada.

La versión gaussiana de la función de base radial mapea los datos desde el espacio de características original a un espacio de características de mayor dimensión, incluso infinitamente dimensional. Es una técnica poderosa, que permite capturar relaciones no lineales en los datos.

- Kernel Sigmoide: este, al igual que los demás tipos de kernel abordados, mide la similitud entre dos vectores de características. Se encuentra definido por la función:

$$K(X_i, X_j) = tanh\,(\alpha * x_i * x_j + \delta)$$

(X_i, X_j) son dos vectores de características en la instancia i-ésima y j-ésima, α es un parámetro que ajusta la pendiente de la función tangente y δ es un parámetro que controla la translación de la función tangente.

El kernel sigmoide es particularmente útil cuando se trata de datos que tienen una estructura no lineal compleja. Es importante tener en cuenta, que el uso de este kernel requiere una cuidadosa sintonización de los parámetros para evitar el sobreajuste.

No existen reglas específicas para determinar qué función de kernel permitirá construir un modelo más preciso. Pero, en general, “el Kernel elegido generalmente no hace una gran diferencia en la precisión resultante”. (Harrigton, 2012)

Los kernels no son exclusivos de las SVM, ya que otros algoritmos de aprendizaje automático también hacen uso de ellos. Un ejemplo común es el kernel de función de sesgo radial, que se emplea en varios contextos de aprendizaje automático para capturar relaciones no lineales entre los datos.

Para transformar un SVM binario a un clasificador multiclase, se emplea el mismo enfoque que con un SVM tradicional, solo que descompone el problema en múltiples clases binarias, que son más fáciles de manejar. Para realizar este proceso, se pueden abordar dos enfoques.

- One-vs-One (Uno contra Uno): En este enfoque, se crean clasificadores para cada par de clases generando todas las posibles combinaciones de las clases en pares. Esto implica que, si hay N cantidad de categorías, se crearán un total de $\frac{N*(N-1)}{2}$ clasificadores binarios. Durante el entrenamiento, cada clasificador binario se entrena utilizando únicamente ejemplos de las dos categorías en cuestión. Por ejemplo, si se tienen las clases A y B, el clasificador se entrena para distinguir los datos que pertenecen a la categoría A o la clase B.

Una vez que se ha entrenado el modelo, el mismo deberá clasificar datos que no pertenecen al conjunto de entrenamiento y, cada clasificador binario, estimará si el conjunto de atributos de entrada se lo puede clasificar en algunas de las dos clases en cuestión. Con relación al ejemplo anterior si esos atributos pertenecen a la categoría A o B. A este proceso se lo denomina “voto”. Una vez que todos los clasificadores han emitido su voto, se realiza un recuento de votos para cada clase en todas las combinaciones de pares. Finalmente, la clase que obtiene la mayoría de los votos, según la determinación de los clasificadores binarios, es la categoría predicha.

- One-vs-All (Uno contra Todos): en este caso, se crea un clasificador binario por cada categoría, en dicho clasificador se contempla la posibilidad que la clase sea positiva o negativa. Continuando con el ejemplo anteriormente mencionado, si se tiene las clases A, B, C, cada clasificador estará formado como (A-positivo, A-negativo), es decir, si el dato pertenece o no a la clase. Al momento de realizar el entrenamiento del modelo se emplean datos específicos, que van a contener casos positivos como negativos y el objetivo es que pueda realizar bien una distinción entre lo que pertenece o no a una clase. Cuando se emplee el modelo para clasificar datos que no son conocidos, cada clasificador binario emitirá su “voto” por si esos datos pertenecen o no a dicha clase. La clase con la cantidad de votos más alta se la considera como la predicción final.

SVM emplea como función de costo, la función Hinge, que mide las discrepancias entre las predicciones del modelo y las etiquetas reales de los datos de entrenamiento. Esta función se encarga de calcular la pérdida o penalización de un modelo según la distancia de los datos a los hiperplanos de separación. La penalización ocurre cuando un dato es mal clasificado o su distancia a un hiperplano es pequeña. La función Hinge contempla los siguientes casos para las penalizaciones:

- Cuando la distancia entre el punto y un hiperplano es mayor o igual a 1, los datos se clasificaron correctamente y la penalización es 0.
- Si la distancia entre el punto y un hiperplano es menor a 1 implica que el punto se encuentra cerca del hiperplano, por lo tanto, existe una penalización, el valor de la misma va a depender en función de qué tan cerca esté del hiperplano (a medida que la distancia disminuye, la pérdida aumenta) y si la distancia es 0 la pérdida es 1.
- Los datos que fueron clasificados incorrectamente recibirán una penalización alta.

La función Hinge busca minimizar la suma total de todas estas penalizaciones o costos mientras ajusta los vectores de pesos y el sesgo de los hiperplanos. Para poder ajustar de forma eficiente los hiperplanos de separación, se utiliza el algoritmo de optimización mínima secuencial (SMO) propuesto por Platt. Es importante destacar, que el algoritmo fue planteado para problemas de clasificación binaria, pero empleando la técnica One-vs-All en conjunto con SMO, puede ser aplicados a SVM multiclase.

El principio de funcionamiento de SMO en las SVM binarias es el mismo que para las SVM multiclase. El problema de optimización se divide en pequeños problemas que puedan ser fáciles de resolver. Abordar el problema de optimización, sin dividirlo en subproblemas, concluirá en el mismo resultado que usando SMO, la única diferencia es que el tiempo de cómputo será mayor.

El algoritmo SMO posee un enfoque iterativo y tiene por objetivo hallar dos $\propto$ (alfas) y un valor para el sesgo o b. Las alfas son coeficientes, que establecen la importancia de los vectores de soporte.

En el proceso iterativo del algoritmo SMO, se seleccionan dos alfas que serán optimizados en cada iteración. Una vez que se ha identificado este par apropiado para una categoría determinada, se procede a incrementar una de las alfas mientras se disminuye la otra. Esta estrategia de actualización de alfas permite ajustar los valores de manera que se maximice el avance hacia la convergencia del algoritmo. Durante el proceso iterativo, se optimizan gradualmente todas las alfas de las clases. Una vez determinados los conjuntos de alfas de cada clase, se pueden calcular fácilmente los pesos w y obtener los hiperplanos de separación.

Es importante destacar, que, para que un conjunto de alfas sea el adecuado, debe cumplir ciertos criterios: ambas alfas deben estar fuera de su límite de margen y que los alfas aún no estén sujetos a restricciones o limitaciones (rango limitado, dependencia de clase, dependencia de la distancia al margen, entre otros).

El proceso de optimización de SMO se repite hasta que se cumple cierto criterio de convergencia o cuando se ha realizado un número máximo de iteraciones. Durante este proceso, las alfas que no están sujetas a restricciones o limitaciones también pueden ser actualizadas para mejorar los ajustes de los hiperplanos de separación.

2.4.3. Redes neuronales

Las redes neuronales se han convertido en el núcleo del aprendizaje profundo, ya que son versátiles y escalables y permiten resolver problemas complejos de aprendizaje automático, como servicios de reconocimiento de voz, clasificación de imágenes, entre otros

problemas. Estas son un modelo de aprendizaje automático inspirado en los mecanismos de aprendizaje de organismos biológicos. *"El sistema nervioso humano contiene células llamadas neuronas. Las neuronas están conectadas entre sí mediante el uso de axones y dendritas, y las regiones de conexión entre axones y dendritas se denominan sinapsis. La fuerza de las conexiones sinápticas a menudo cambia en respuesta a estímulos externos. Este cambio es la forma en que se produce el aprendizaje en los organismos vivos."* (Aggarwual, Charu. 2019). Las redes neuronales artificiales intentan imitar este comportamiento y están conformadas por unidades de cálculos denominadas neuronas.

Existen diferentes arquitecturas de redes neuronales artificiales, la más simple de ellas es el perceptrón, esta se encuentra conformada por una única capa de entrada y una salida (ver la imagen izquierda de la figura 7). La capa de entrada, está compuesta por neuronas que no realizan ningún tipo de operación, dado que solo propagan las características a la capa de salida. Cada conexión de las neuronas de la capa de entrada a la salida tiene asociadas pesos sinápticos y es la última capa la que se encarga de multiplicar dichos pesos con las características para luego aplicar una sumatoria. Finalmente, se emplea una función escalonada para determinar el resultado. Generalmente, se emplean la siguiente función escalonada:

$$heaviside(z) = \begin{cases} 0 \; if \; z < 0 \\ 1 \; if \; z \geq 0 \end{cases}$$

Donde z es el resultado de la sumatoria ponderada de la multiplicación de los atributos con los pesos. Luego de aplicar la función $heaviside(z)$, la neurona se activa dependiendo si el resultado excede el umbral.

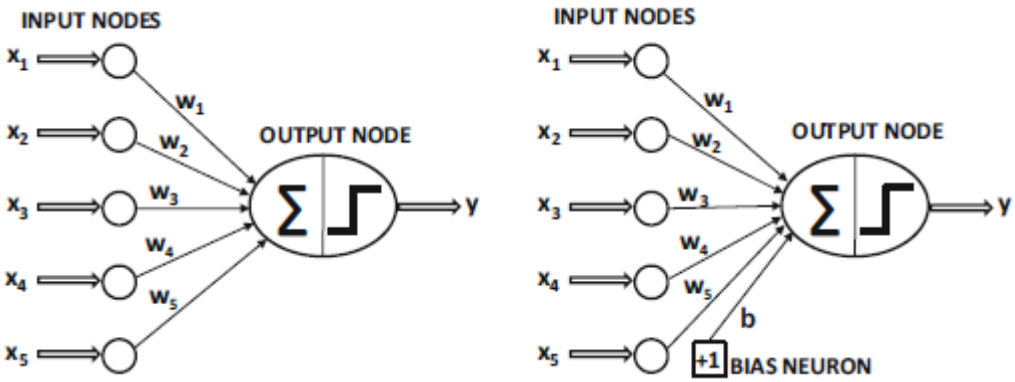


Figura 7. Perceptrón.
Fuente: Aggarwual C. (2018) "Introduction" en *Neural Networks and Deep Learning*. Capítulo 1.2, p 5.

La arquitectura anteriormente mencionada es ineficiente cuando se trabaja con datos altamente desequilibrados, ya que la media de la predicción no se centra en 0, por lo tanto,

el perceptrón realizará predicciones incorrectas en la clase minoritaria. Para suplir esta situación, se adiciona una neurona de sesgo (ver la imagen derecha de la figura 7) que siempre transmite un valor de 1 al nodo de salida, y el peso de la conexión entre la neurona de sesgo y el nodo de salida determina el valor del sesgo. De esta manera, el modelo puede aprender a ajustar su predicción ante situaciones de clases desbalanceadas.

El aprendizaje del perceptrón se basa en recorrer repetidamente todas las muestras de entrenamiento de forma aleatoria y realizar las predicciones. Como dichas muestras están conformadas por pares entrada-salida, es decir, que se conoce el resultado que se espera para determinadas entradas, se puede saber si una predicción es correcta o no, y luego ajustar los pesos en base a los errores de predicción. Este ajuste se realiza de forma sucesiva y se puede lograr la convergencia del perceptrón, es decir, se minimiza el número de clasificaciones incorrectas.

Es importante destacar que como cada neurona de salida es lineal, el perceptrón es bueno para clasificar datos que son linealmente separables, caso contrario son incapaces de aprender patrones complejos, por lo tanto, generan resultados ineficientes. Ante esta situación se desarrollaron redes neuronales más complejas que intentan abordar estas problemáticas. Algunos ejemplos son los siguientes: redes neuronales artificiales (perceptrón multicapa), redes neuronales convolucionales o redes neuronales profundas, etc. En el presente trabajo solo se abordarán las redes neuronales artificiales, debido a que son las empleadas para el desarrollo de la práctica.

Una red neuronal artificial está compuesta por una serie de capas intermedias, denominadas ocultas, entre la capa de entrada y la de salida. Cuando todas las neuronas de cada capa se encuentran conectadas a cada neurona de la capa anterior se denomina que la red es densa (ver figura 8). Al igual que en el perceptrón, la capa de entrada solo se encarga de propagar las muestras a la capa siguiente, y las capas sucesivas se alimentan entre sí en la dirección de avance desde la entrada.

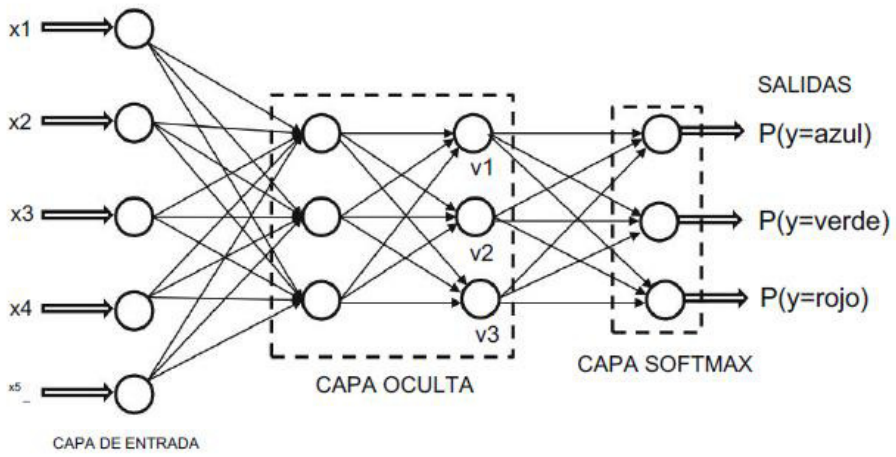


Figura 8. Red Neuronal Densa.
Fuente: Aggarwual C. (2018) "Introduction" en *Neural Networks and Deep Learning*. Capítulo 1.2, p 14.

Cada capa oculta contiene una matriz de pesos asociada que determina cómo las neuronas de esa capa están conectadas a la de la siguiente. La primera capa oculta, posee una matriz de pesos llamada $W1$ de tamaño $d * p1$, donde d es la dimensión de los datos de entrada y p1 es la cantidad de neuronas en esa capa. Del mismo modo, las subsiguientes capas ocultas también tienen una matriz de pesos llamada Wr (donde r indica el número de la capa) de tamaño $pr * pr + 1$, donde pr es la cantidad de neuronas en la r-ésima capa y $pr + 1$ es la cantidad de neuronas en la capa siguiente. Por último, la matriz que se encarga de conectar la última capa oculta con la última capa de la red neuronal se denomina $Wk + 1$ y tiene un tamaño de $pr + 1 * o$ donde o indica el número de neuronas en esa capa.

Por cada capa se define la función de activación que las neuronas que la componen deben emplear, esta función se aplica a la suma ponderada de las multiplicaciones de los pesos sinápticos con cada una de las entradas de la neurona, el resultado se propagará a la neurona de la capa siguiente. El resultado de esta transformación permite limitar el rango de amplitud permisible de la salida a un valor finito que oscila entre [0 y 1] o [-1 y 1], es decir, que la salida indicará la probabilidad de que la entrada corresponda a una determinada clase. De esta forma se podrán ajustar de forma gradual los pesos, permitiendo un entrenamiento más eficaz.

Existen diferentes tipos de funciones de activación, la distribución de las mismas se puede visualizar en la figura 9. A continuación se describen algunas de ellas:

- Función Sigmoide: los valores de salida están comprendidos dentro de un rango que va de 0 a 1. Esta función, tiene una forma similar a la función heaviside(z), pero se encuentra suavizada, es decir, que representa un continuo de posibles grados de

activación, en vez de una salida binaria. La función Sigmoide indicará cuán cerca están determinados atributos de pertenecer a una clase. Se encuentra representada por la siguiente ecuación:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

x representa la entrada de la función sigmoide. Cuando x toma valores muy grandes, ya sea positivos o negativos, la función exponencial que se encuentra en el denominador se vuelve muy grande o pequeña, generando que $\sigma(x)$ tienda a saturarse en los extremos (0,1). Si esto ocurre, los gradientes asociados a la retropropagación se vuelven muy pequeños, produciendo estancamientos en el aprendizaje.

- Función tangente hiperbólica (tanh): los valores de salida están comprendidos dentro de un rango que va de -1 a 1. Esto hace que la salida de cada capa esté aproximadamente centrada alrededor de 0 al comienzo del entrenamiento, esto puede ayudar a una convergencia más rápida. Esta función es una reescalada de la función sigmoide y su utilización es adecuada cuando se emplea un modelo lineal. Al igual que con la función presenta problemas de "saturación" cuando los valores de salida se encuentran cerca de los extremos (-1, 1), ya que resulta en gradientes muy pequeños, generando dificultades en el aprendizaje. La función tangente hiperbólica se la representa con la siguiente ecuación:

$$2\sigma(2z) - 1$$

- Función de unidad lineal rectificada (ReLU): es una función continua cuya pendiente cambia abruptamente en $Z = 0$, lo que puede hacer que el Descenso de Gradiente salte alrededor, y su derivada es 0 para $Z < 0$. Cabe destacar, que el hecho de no tener un valor máximo de salida ayuda a reducir algunos problemas durante el Descenso de Gradiente. Sin embargo, se introduce un nuevo problema, que es cuando las entradas son negativas, en este caso el gradiente se desvanece, si en el entrenamiento se produce esta condición, las neuronas quedan "muertas". Sin embargo, la función ReLU es más rápida de entrenar porque su gradiente es eficiente de calcular cuando su entrada no es negativa.

$$\max(0, Z)$$

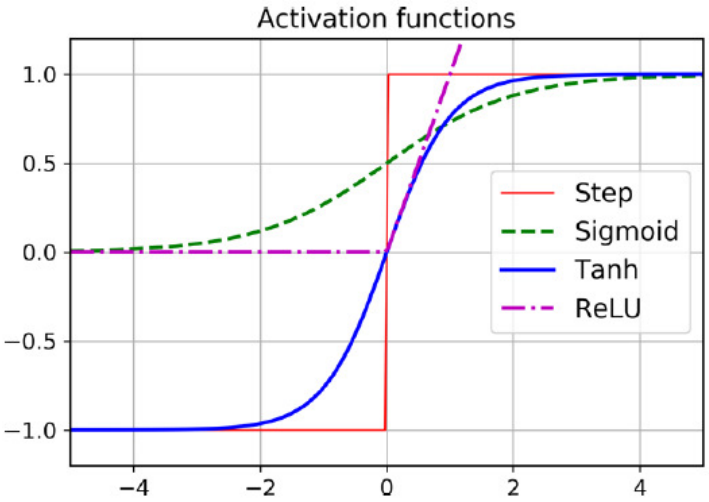


Figura 9. Figura 9. Distribución de las funciones de activación sigmoide (color verde), Tanh (color azul), Relu (color morado), escalonada (color rojo).
Fuente: Extraído de Gerón A. (2017) "Introduction to Artificial Neural Networks with Keras" en *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Capítulo The Multilayer Perceptron and Backpropagation, p 292

Las funciones de activación en las capas ocultas son sumamente importantes debido a que con ellas se obtiene gran parte del poder aprendizaje en las redes neuronales. Es por ello, que se realiza la composición repetida de determinadas funciones de activación. Si estas funciones son solamente lineales, el modelo no podrá resolver problemas complejos y aunque la red neuronal esté conformada por varias capas, será como tener un perceptrón con una única capa. Sin embargo, si se realiza la composición de un gran número de funciones no lineales, se podrá aproximar casi cualquier función, este enfoque requiere de un número grande de unidades (es decir, parámetros) en la red. Consecuentemente, aumenta la capacidad de la red, lo que provoca sobreajuste a menos que el conjunto de datos sea extremadamente grande.

Es importante resaltar, que en la capa de salida para el caso particular de los problemas de clasificación multiclase, se utiliza la función SoftMax, ya que transforma las salidas en una distribución de probabilidad de pertenecer a cada una de las posibles clases. Para ello, se calcula en base a una muestra de entrada, la evidencia de pertenecer a cada una de las clases, luego estas, se deben convertir en probabilidades sumando las salidas de todas las neuronas de la última capa oculta debe y esta debe ser uno. Para lograr esto, SoftMax usa el valor exponencial de las evidencias calculadas y las normaliza de modo que el resultado sea uno, formando una distribución de probabilidad. Por lo tanto, la probabilidad de pertenecer a una clase se exprese con la siguiente ecuación:

$$\text{SoftMax}_i = \frac{e^{evidencias_i}}{\sum_j e^{evidencias_j}}$$

Una buena predicción tendrá una sola entrada con un valor cercano a 1, mientras que las entradas restantes serán cercanas a 0.

El objetivo de una red neuronal artificial, empleada para la clasificación, es aproximar una función g, que no es conocida en base a los datos de entrada, ya que se considera que los atributos guardan una relación con la categoría a la que pertenecen. Por lo tanto, se busca aproximar g por medio de la composición de las funciones de activación de las capas que componen la red (que llamaremos f), a partir del conocimiento de una cantidad limitada de puntos de g. Luego se espera que f se comporte de manera similar a g para otras muestras análogas. Esta aproximación, busca minimizar la función de costos, para ello se ajustan los pesos de la red para disminuir los errores. En los problemas de clasificación multiclase se emplea la función de costos entropía cruzada categórica, que se la representa con la siguiente ecuación:

$$H(p,q) = -\sum_{x} p(x) \log \mathrm{q}(x)$$

Donde $p(x)$ representa la probabilidad real que la muestra pertenezca a una clase x y $\mathrm{q}(x)$ es la probabilidad predicha por el modelo de que la muestra pertenezca a la clase x.

Existen diversos métodos para reducir la función de costos, como el descenso de gradiente y el descenso de gradiente estocástico. Este último tiene la finalidad de determinar la tasa de aprendizaje, que es la velocidad con la que aprende la red. Una tasa de aprendizaje demasiado alta, puede ocasionar que la red no pueda disminuir el costo de entrenamiento más allá de cierto punto. Por otro lado, una tasa demasiado baja genera que la función de costo disminuya lentamente, ralentizando el aprendizaje. El objetivo es lograr un equilibrio entre una disminución significativa del costo, pero sin perder detalles importantes de las muestras que permitirán que la red tenga una buena capacidad para generalizar y que lo haga a una velocidad razonable, contemplando el tamaño de las muestras.

Por otra parte, durante el proceso de entrenamiento de la red se busca optimizar alguna medida, que se denominará P, que representa el rendimiento. P y que depende del conjunto de prueba, no del entrenamiento: por lo tanto, no se conoce su distribución. Sin embargo, se busca optimizar P de forma indirecta, para ello, se recurre a otra función de costos denominada L, con el objetivo de logar una mejoría en P. De esta forma, se intenta reducir el error de generalización del modelo, que se lo puede representar como:

$$\mathrm{R}(\theta) = E_p[L(f(x,\theta), y)]$$

Donde P es la distribución de los pares (x, y), que son los atributos y la categoría a la que pertenecen, L es la función, que cuantifica la pérdida para cada par $f(x, \theta)$ y $f(x, \theta)$ es la salida de la red. Generalmente, como no se conoce P, se minimiza el costo del conjunto de entrenamiento en vez de hacerlo sobre todo el conjunto. A esto, se lo denomina reducir el riesgo empírico y se lo representa con la siguiente ecuación:

$$J(\theta) = E_p[L(f(x,\theta), y)] = \frac{1}{n}\sum_{k+1}^{n} L(f(x_k,\theta), y_k)$$

Con este enfoque se espera poder reducir $\mathrm{R}(\theta)$ de forma significativa. Sin embargo, es propenso al overfitting, ya que los modelos complejos pueden simplemente memorizar el conjunto de entrenamiento.

Para los casos de clasificación se emplea el optimizador Adam (Adaptive Moment Estimation), que calcula las tasas de aprendizaje por cada parámetro. Para ello guarda una media móvil exponencial de los gradientes al cuadrado, además de guardar una media móvil de los gradientes sin el cuadrado. Se los representa de la siguiente forma:

$$mt = \propto m_{t-1} + (1{+}\propto) g_t$$

$$vt = \beta v_{t-1} + (1+\beta) g_t^2$$

mt es un estimador del primer momento de los gradientes. En el primer momento se actualiza cada parámetro en cada paso del entrenamiento, además sirve para tener una noción general de la dirección de los gradientes, indicando hacia dónde se mueven.

vt es un estimador del segundo momento, que es la estimación del promedio de los gradientes al cuadrado. Este se actualiza en cada paso de entrenamiento para cada parámetro y ayuda a tener una noción de qué tan bruscos son los cambios.

Dado que mt y vt son inicializados como vectores cero, esto ocasiona que estén sesgados hacia el cero, particularmente durante los pasos iniciales, cuando las tasas de decaimiento son pequeñas. Para contrarrestar esto, se introducen unos estimadores corregidos y para la actualización de los parámetros se emplea la siguiente función:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Estimador de la media móvil de los gradientes sin el cuadrado

$$\hat{m}_t = \frac{mt}{1 - \propto^t}$$

Estimador de la media móvil exponencial de los gradientes al cuadrado

$$\hat{v}_t = \frac{vt}{1-\beta^t}$$

Es importante destacar que Adam no emplea una tasa de aprendizaje constante para todos los parámetros, sino que adapta la tasa de aprendizaje para cada parámetro de forma individual en función de la estimación del primero momento y segundo momento ($\hat{m}_t$, $\hat{v}_t$) y de la magnitud del gradiente. Esto permite que el modelo se ajuste de forma más eficaz en los datos de entrenamiento.

Tras exponer las funciones de activación que pueden emplear las neuronas tanto en las capas ocultas como en la capa de salida, así como la elección de la función de pérdida y el optimizador para minimizar dicha función de costos, es fundamental abordar el ajuste de los pesos durante el entrenamiento de la red. Para ello, se emplea el algoritmo de retropropagación, el cual utiliza un conjunto de mini lotes e itera el conjunto de entrenamiento completo varias veces, cada iteración o paso se lo denomina época. Durante cada época, se envía cada mini lote a la capa de entrada de la red, que lo direcciona a la primera capa oculta. Las neuronas al aplicar sus funciones de activación, calculan los resultados para cada muestra del mini lote. Estos resultados se propagan a la siguiente capa, repitiéndose el proceso hasta alcanzar la capa de salida, que genera la predicción. No obstante, todos los resultados intermedios se conservan ya que son necesarios para el ajuste de los pesos sinápticos.

Una vez, que se conoce la predicción se calcula la función de costo, que mide la discrepancia entre la clase esperada respecto a la clase predicha. Posteriormente, se determina la contribución de cada conexión de salida al error aplicando la regla de la cadena, logrando así un paso rápido y preciso. El algoritmo luego recorre cada capa en sentido inverso, comenzando por la última capa, y calcula los gradientes de la función de pérdida con respecto a los pesos de la red. Estos gradientes indican la dirección y magnitud en la que cada peso debe ajustarse para reducir la pérdida. Finalmente, los pesos de las conexiones se ajustan para disminuir los errores de la red, y en el caso específico de la red desarrollada para esta práctica, se emplea el optimizador Adam para llevar a cabo estos ajustes.

2.5. Dataset

El dataset o conjunto de datos es utilizado para entrenar, validar y evaluar modelos de Machine Learning. En el caso específico del aprendizaje automático, este conjunto de datos contiene ejemplos acompañados de sus respectivas etiquetas. La subdivisión en un

conjunto de validación se emplea porque entrenar un modelo y validar el mismo con los datos de entrenamiento es un error metodológico, ya que el modelo no tendrá la capacidad de generalizar y de realizar predicciones precisas en datos no conocidos y, esta situación se la denomina sobreajuste. Por lo tanto, para el entrenamiento y validación o evaluación se puede emplear el método de retención de datos, que consiste en subdividir al conjunto de datos en al menos dos grupos ver figura 10, y en ciertos casos, tres grupos estáticos.

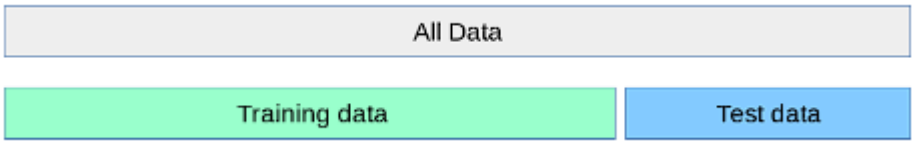


Figura 10. División del conjunto de datos en datos de entrenamiento y prueba
Fuente: Extraído de https://scikit-learn.org/stable/modules/cross_validation.html

En el caso que se aborde el enfoque de subdividir el conjunto de datos en dos grupos, el grupo principal es destinado a los datos de entrenamiento, los cuales deben mantener un equilibrio cuidadoso para garantizar una distribución similar de etiquetas. Un desequilibrio en esta distribución podría llevar a deficiencias en el entrenamiento del modelo, afectando su capacidad para reconocer diversas etiquetas. Al segundo grupo, conocido como conjunto de prueba, se lo emplea para evaluar la capacidad de generalizar del modelo.

Si se aborda el enfoque se subdividir el conjunto en tres grupos, se emplea la misma técnica de tener un grupo principal destinado a los datos de entrenamiento y luego se añade el conjunto de validación. Este nuevo subconjunto es empleado para evaluar y ajustar los hiperparámetros[1] del modelo antes de evaluar su rendimiento final. Posteriormente, se emplea el conjunto de pruebas, que al igual que en el enfoque anterior, brindará información acerca de la eficiencia del modelo.

No existe una regla que indique qué porcentaje de los datos deben ser empleados para el entrenamiento, validación y pruebas. Sin embargo, si se utiliza el primer enfoque, por lo general, se destina un 80% del total del dataset al conjunto de entrenamiento y el 20% restante para pruebas. Por otro lado, si se emplea el segundo enfoque, se puede mantener el 80% del total de los datos para entrenamiento, mientras que se usa un 10% para validación, y el 10% restante para pruebas. Otro enfoque empleado es reducir el conjunto de entrenamiento, es decir, se empleará el 70% para entrenamiento, 10% para validación y el restante para pruebas. La subdivisión en el dataset se realiza de forma manual y a discreción del desarrollador. Reducir el tamaño del conjunto de entrenamiento puede generar que el

[1] Hiperparámetros: son un conjunto de variables que se configuran por un desarrollador y definen la arquitectura, la regularización y otros aspectos del comportamiento de un modelo.

modelo pierda eficiencia en la clasificación de datos no conocidos, lo que significa que podría tener dificultades para generalizar.

Por otra parte, es importante resaltar que la validación es un paso fundamental en la selección del modelo más adecuado al problema que se está tratando, es por ello, que se emplea la técnica de cross-validation (validación cruzada o CV), ya que es robusta y precisa. Esta consiste en generar conjuntos de entrenamiento y validación para evaluar la capacidad de generalización de un algoritmo para un conjunto de datos específico y, con los resultados obtenidos, se puede determinar si es necesario realizar ajustes en los hiperámetros del modelo.

Una de las técnicas de validación cruzada es K-Fold, es iterativa y consiste en dividir el set de datos en k subconjuntos ver figura 11. En cada iteración, se emplea uno de los subconjuntos, como conjunto de validación para evaluar el modelo, y los k-1 subconjuntos restantes se los emplea para ajustar el modelo. Este proceso se lo repite k veces. Se debe destacar que el valor de k suele oscilar entre 5 y 10, y solo se eligen valores grandes de k cuando el set de datos es grande. Sin embargo, es fundamental tener en cuenta, que esto puede aumentar el riesgo de sobreajuste en los datos.

Las principales ventajas de emplear cross-validation son las siguientes:

- Elimina la aleatoriedad que introduce el método de retención, en el subconjunto de validación, ya que garantiza que los pliegues de entrenamiento y validación estén conformados por ejemplos fáciles de clasificar, que son aquellos cuyos atributos son claramente indicativos de una clase en particular y ejemplos complejos que pueden tener características que se superponen entre las diferentes clases. De esta forma, el modelo tendrá la capacidad para generalizar bien todas las muestras del conjunto de datos.
- Proporciona una aproximación de cómo podría comportarse el modelo en los peores y mejores escenarios al aplicarse nuevos datos.

La principal desventaja de la validación cruzada es el aumento del costo computacional. Dado que se está entrenando k modelos en lugar de un solo modelo, la validación cruzada será aproximadamente k veces más lenta que hacer una única división de los datos.

Una vez concluida la etapa de validación y ajuste, se debe entrenar el modelo con todo el conjunto de datos y luego probarlo con datos no conocidos para evaluar su desempeño.

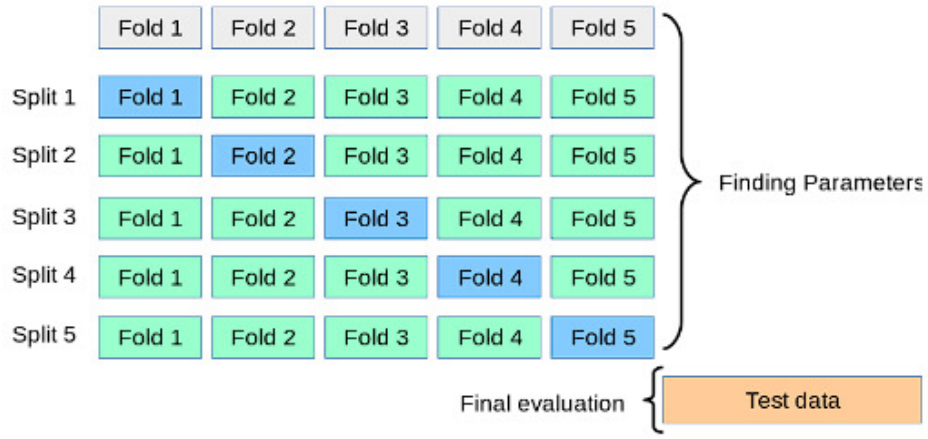


Figura 11.División del conjunto de datos con la técnica K-Fold.
Fuente: Extraído de https://scikit-learn.org/stable/modules/cross_validation.html

2.6. Métricas

Para evaluar el rendimiento de los modelos se emplean diferentes métricas y herramientas. Para la selección de las mismas se debe tener en cuenta el objetivo final de la aplicación, ya que generalmente, no solo se busca que el modelo tenga la capacidad de realizar predicciones precisas, sino también, se puedan utilizar dichas predicciones para la toma de decisiones. En el proceso de selección de una métrica se debe tener en cuenta las medidas de desempeño. Esto hace referencia a cómo se define la evaluación de la precisión y si se valora un aspecto del rendimiento más que otro. En la estimación de errores, una vez que se seleccionó una medida de desempeño, se busca determinar de forma imparcial, los costos asimétricos de una clasificación errónea. A este aspecto se lo tiene en cuenta cuando clasificar erróneamente una clase es considerado más grave que en otras.

Por otro lado, es importante resaltar, que dependiendo el algoritmo se emplean determinadas herramientas o métricas, sin embargo, en este apartado solo se abordan las métricas que son empleadas comúnmente para los modelos de clasificación; no obstante, esto no implica que estas métricas no puedan ser empleadas en otros algoritmos.

2.6.1. Matriz de confusión

Es uno de los métodos más completos para representar los resultados de la evaluación de un modelo. Debido a que muestra un desglose detallado de las clasificaciones correctas e incorrectas por cada clase, ver figura 12.

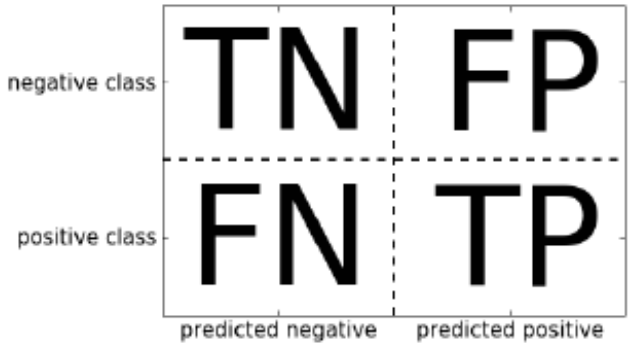


Figura 12.. Representación de matriz de confusión para un modelo de clasificación binaria.
Fuente: Müller A. & Guido S. (2016) "Model Evaluation and Improvement" en *Introduction to Machine Learning with Python. A guide for data scientists*. Capítulo Metrics for Binary Classification, p 281.

La matriz se encuentra conformada por filas que representan las categorías reales del conjunto de datos, mientras que las columnas representan las categorías predichas por el modelo.

2.6.2. Accuracy

Una forma de resumir el resultado de la matriz de confusión es mediante el cálculo de Accuracy, que se lo puede expresar de la siguiente forma:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- TP verdaderos positivos: son el número de muestras que fueron clasificados correctamente como positivas por el modelo.
- TN verdaderos negativos: son el número de muestras clasificadas como falsas de forma correcta
- FP: falsos positivos: son el número de muestras clasificadas como positivas de forma errónea, ya que eran negativas.
- FN falsos negativos: son el número de muestras clasificadas como falsas erróneamente, ya que eran positivas

La métrica Accuracy representa el total de muestras clasificadas de forma correcta sobre el número de muestras total.

Esta métrica es útil cuando los errores de clasificación tienen el mismo orden de importancia en todas las clases. Por otro lado, es recomendable emplearla solo si se tiene una distribución de clases balanceada en el conjunto de datos, es decir, un número similar de muestras en cada categoría. Esto se debe a que esta métrica mide el rendimiento general del modelo y no por clase, por lo tanto, si en las muestras existen clases predominante, el

modelo podrá clasificarlas correctamente dando como resultado un porcentaje alto de precisión, a pesar de que el modelo presente problemas para clasificar correctamente las clases minoritarias.

2.6.3. Precision

Esta métrica tiene por objetivo determinar de todos los elementos que se clasificaron como positivos, cuáles realmente lo son. Por lo tanto, es empleada cuando se busca medir el rendimiento del modelo limitando el número de falsos positivos. Se la puede expresar de la siguiente forma:

$$Precision = \frac{TP}{TP + FP}$$

- TP verdaderos positivos: son el número de muestras que fueron clasificados correctamente como positivas por el modelo.
- FN falsos negativos: son el número de muestras clasificadas como falsas erróneamente, ya que eran positivas

2.6.4. Recall

Esta métrica tiene por objetivo determinar cuántos elementos fueron clasificados correctamente por el modelo. Por lo tanto, es empleada cuando se intenta evitar los falsos negativos. Se la puede expresar de la siguiente forma:

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

"*Existe un equilibrio entre optimizar la métrica recall y optimizar la precision. Se puede obtener trivialmente un recall perfecto si se predice que todas las muestras pertenecen a la clase positiva: no habrá falsos negativos ni tampoco verdaderos negativos. Sin embargo, predecir todas las muestras como positivas dará como resultado muchos falsos positivos y, por lo tanto, la precision será muy baja. Por otro lado, si encuentra un modelo que predice solo el punto de datos del que está más seguro como positivo y el resto como negativo, entonces la precisión será perfecta (suponiendo que este punto de datos sea positivo), pero la recuperación será muy mala*" (Andreas C. Müller & Sarah Guido, 2016).

2.6.5. F1-Score

Tanto la precision como recall son métricas que proporcionan información importante del rendimiento del modelo, sin embargo, la métrica F1-Score proporciona una visión más

completa, ya que contempla a ambas para medir el rendimiento. Esta es representada por la siguiente ecuación.

$$f1 - score = 2 * \frac{precision * recall}{presicion + recall}$$

Cuando se quiere evaluar el rendimiento general del modelo empleando $f1 - score$, primero se calcula un f-score binario por cada clase, considerando dicha clase como positiva y las restantes negativas. Luego, los resultados obtenidos por cada categoría se promedian empleando una las siguientes estrategias:

- Promedio “macro”: no pondera las clases, ya que le otorga el mismo orden de importancia o peso a todas las categorías, independientemente de su tamaño.
- Promedio “micro”: calcula el total de falsos positivos, falsos negativos, y verdaderos positivos de todas las clases, luego computa la precision y recall y partir de los resultados de estos últimos, calcula el $f1 - score$.
- Promedio “weighted”: primero, se calcula el $f1 - score$ de cada clase, luego lo multiplica por un valor ponderado de dicha clase, que está relacionado con el número de muestras de dicha categoría. Las clases con más muestras tienen una mayor influencia en el cálculo general. Finalmente, se calcula la media de los $f1 - score$ y este valor es el que representa el rendimiento general del modelo.

Esta métrica es recomendable emplear cuando se trabaja con un problema de clasificación multiclase y el conjunto de datos se encuentra desbalanceado.

2.7. Arquitecturas Android

Existen diversas arquitecturas que pueden ser implementadas en el desarrollo de una aplicación Android, algunas de ellas son Model View Controller (MVC), Model View Presenter (MVP), Model-View-View-Model (MVVM). La diferencia entre ellas radica en la forma en que interactúan cada uno de sus componentes. Por otro lado, es importante destacar que la arquitectura recomendada por Android es MVVM y, esta se estructura en al menos dos módulos básicos (ver figura 13) que son los siguientes: la capa de interfaz de usuario y la capa de datos. Opcionalmente, se puede implementar la capa de dominio, que es la encargada de encapsular la lógica del negocio. Estos son algunos de los módulos que pueden definirse; no obstante, a medida que una aplicación se vuelve más compleja, es necesario definir otros para evitar la duplicidad de código.

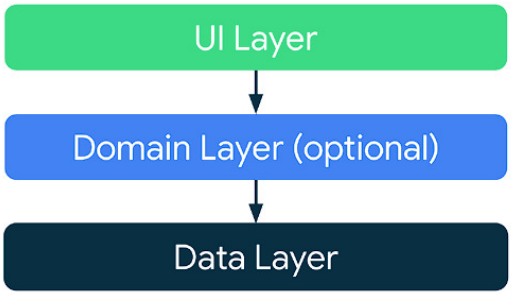


Figura 13. Arquitectura básica de una aplicación para dispositivos.
Fuente: Extraído de https://developer.android.com/jetpack/guide?hl=es-419

La interacción entre los componentes básicos de una aplicación con arquitectura mvvm se puede visualizar en la figura 14. Cada componente solo depende del componente que está un nivel más abajo, solo las clases repository, podrían depender de más de una componente, ya que la aplicación emplearía varias fuentes de datos.

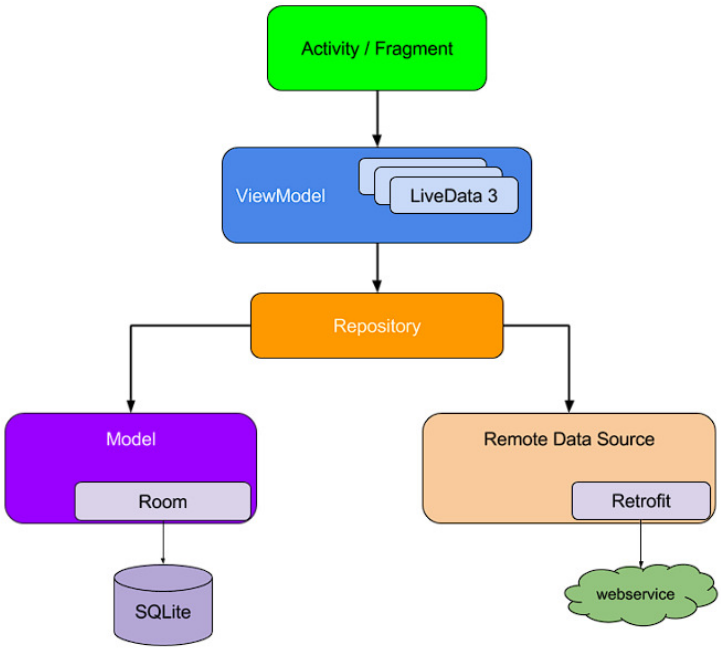


Figura 14.. Interacción entre los componentes de una aplicación.
Fuente: Extraído de https://developer.android.com/jetpack/guide?hl=es-419

2.7.1 Capa de interfaz de usuario

La capa de interfaz de usuario (UI Layer) hace referencia a los elementos que componen la interfaz de usuario de una aplicación. Estos elementos tienen la responsabilidad de presentar la información de manera que sea comprensible y permita la interacción del usuario con ellos. Para lograr esta funcionalidad, se utilizan contenedores de estado que son los

encargados de administrar el estado y de propagarlos. Para administrar los cambios de estado de la UI es recomendable emplear clases que extiendan de ViewModel[2].

En el proceso de interacción usuario-aplicación, el usuario interactúa con los elementos de la UI, y esta interacción produce un evento que fluye hasta el contenedor de estado o ViewModel, en el caso definido en esta práctica. El contendor se encargará de llamar al método de la clase encargada de realizar la acción solicitada por el usuario y, una vez obtenido el resultado, se producirá un cambio de estado. Estos cambios de estado, por lo general, se realizan a contendores de datos específicos, ya que pueden ser observados por la UI y partir de dichos cambios renderizar la nueva información y presentarla al usuario.

La comunicación entre estos componentes sigue un flujo de datos unidireccional, ya que se intenta separar el lugar donde se producen los cambios de estado, donde se transforman y el lugar donde se muestran, debido a que se busca cumplir con el principio de responsabilidad única y que la UI solo se encargue de renderizar la información. Es importante aclarar, que en ocasiones tanto el lugar donde se produce el cambio de estado como el consumidor son el mismo. Un ejemplo de esto es cuando un usuario hace clic sobre una imagen para obtener más información, en este caso, el cambio de estado es el clic en la imagen, el ViewModel se encargará de llamar a la lógica de negocio de la aplicación para buscar la información y una vez obtenida cambiar el estado para que finalmente sea la UI la que se encargará de renderizar la información y presentarla nuevamente al usuario.

2.7.2. Capa de dominio.

Como se mencionó con anterioridad, esta capa es opcional, ya que depende de la complejidad de la aplicación y se encuentra entre la interfaz de usuario y capa de datos, ya que es la encargada de comunicarse con la UI y depende de las clases repository, para proporcionar la información solicitada por la UI.

Su funcionalidad principal es encapsular la lógica del negocio, es decir, que para determinadas funcionalidades en la aplicación que impliquen cierta complejidad, lo correcto es crear un caso de uso que contenga toda esta lógica. El objetivo es que la UI solo debe encargarse de presentar los datos al usuario y capturar los eventos que él produzca a partir de la interacción con la aplicación y la capa de la UI solo se debe realizar operaciones concernientes a los datos. Es importante mencionar, que los casos de uso pueden ser empleados por más de una clase, por lo tanto, su implementación mejora la escalabilidad y

[2] ViewModel: Definen la lógica que se aplica a los eventos en la app y, como resultado, producen un estado actualizado.

la mantenibilidad del código, debido a que las correcciones o mejoras solo se deben realizar en una única clase.

Por otro lado, al implementar la capa de dominio se evita la duplicidad de código, facilita la implementación de test unitarios, además de evitar clases grandes que tengan muchas responsabilidades, ya que esto infringe el principio de responsabilidad única; por lo tanto, esta capa favorece la división de responsabilidades al igual que la reutilización de código.

2.7.3. Capa de datos

La capa de datos desempeña un papel crucial en la aplicación, ya que se encarga de la persistencia y de la gestión de los datos utilizados en la aplicación. Esta capa puede estar compuesta por una o varias fuentes de datos, como archivos locales, fuentes de red, bases de datos locales o Cloud. La información proveniente de cada fuente de datos es modelada en clases o data class, dependiendo del lenguaje de programación empleado. Esto permite un manejo más flexible de la estructura de los datos, ya que solo se consideran aquellos parámetros que son de interés.

Por otro lado, se definen clases repository que están vinculadas a uno o más DataSource. Estas clases tienen como principal objetivo exponer los datos al resto de la aplicación, y que estos sean inmutables, lo que significa que otras clases únicamente pueden consultar, pero no pueden modificar los datos directamente. De esta forma se garantiza la coherencia y la integridad de los datos a lo largo de la aplicación. Además, estas clases se encargan de centralizar la gestión de los datos convirtiéndose en el único punto de acceso para la consulta, actualización, inserción o eliminación de los datos, logrando que la fuente de datos se abstraiga del resto de la aplicación, ya que es incorrecto que se pueda acceder al DataSource desde la UI o desde la capa de dominio. Finalmente, esta clase también se encarga de la resolución de conflictos entre las diferentes fuentes de datos.

3. Desarrollo del modelo

3.1. Herramientas empleadas

En esta sección se describen las herramientas empleadas en el desarrollo de la Práctica Profesional Supervisada, junto a una breve explicación de cada una de ellas.

3.1.1. Entornos

Para llevar a cabo el desarrollo del modelo de aprendizaje automático se empleó Google Colab o Colaboratory, el cual es un entorno de desarrollo en línea que permite a los programadores escribir y ejecutar código Python en el navegador. Se trata de un servicio basado en el proyecto Jupyter Notebook. Además, Colaboratory ofrece acceso gratuito a recursos de cómputo en la nube, incluidas unidades de procesamiento gráfico (GPUs) y unidades de procesamiento tensorial. Es importante mencionar que los recursos que ofrece no son ilimitados, ya que se busca ofrecer servicios a un gran número de usuarios.

Tanto Colaboratory (que es una herramienta Cloud) y Android Studio pueden ser empleados en los Sistemas Operativos Windows, Mac y Linux. Sin embargo, todo el proyecto fue desarrollado en un equipo con Windows 10. El equipo cuenta con las características expuestas en la figura 15.

Figura 15. Características del equipo de trabajo.
Fuente: elaboración propia basada en los componentes del equipo donde se desarrolló la práctica.

3.1.2. Software y librerías

Para la construcción del modelo se empleó el lenguaje de programación Python. Se trata de un lenguaje interpretado, que es ampliamente utilizado en el aprendizaje automático e inteligencia artificial, así como también en la ciencia de datos, entre otros casos de uso.

Es importante destacar que Python cuenta con una gran variedad de librerías que sirven para la creación de modelos de aprendizaje automático. En este trabajo se emplearon las siguientes:

- Pandas
- Tensor Flow
- Tensor Flow Lite
- Keras
- Numpy
- Scikit-learn
- keras-tuner

3.1.3. Dataset Empleado

Se empleó el conjunto de datos "Crop Recommendation Dataset" para el entrenamiento y validación del modelo. Este conjunto de datos está disponible de manera gratuita en la plataforma Kaggle, la cual ofrece una amplia variedad de datasets.

Crop Recommendation Dataset contiene información relevante sobre el suelo y se construyó utilizando datos de cultivos en la región de la India (ver figura 16). Entre los parámetros de mayor interés que proporciona este dataset se encuentran:

- N: indica la relación de contenido de nitrógeno en el suelo.
- P: indica la relación de contenido de fósforo en el suelo.
- K: indica la relación de contenido de potasio en el suelo.
- Temperature: indica la temperatura del suelo expresada en grados Celsius.
- Humidity: indica la humedad relativa del suelo expresada en porcentaje.
- pH: indica el valor de pH del suelo.
- Rainfall: indica la precipitación expresada en mililitros.

Cada conjunto de valores del suelo está asociado a una categoría específica, que representa el cultivo correspondiente. Las categorías disponibles en el dataset son las siguientes: arroz, maíz, garbanzo, frijol rojo, guisantes, frijol de rocío, frijol mungo, vigna mungo, lenteja, granada, plátano, mango, sandía, melón, manzana, naranja, papaya, coco, algodón, yute, café, uva.

N	P	K	temperature	humidity	ph	rainfall	label
90	42	43	20.879744	82.002744	6.502985	202.935536	rice
85	58	41	21.770462	80.319644	7.038096	226.655537	rice
60	55	44	23.004459	82.320763	7.840207	263.964248	rice
74	35	40	26.491096	80.158363	6.980401	242.864034	rice
78	42	42	20.130175	81.604873	7.628473	262.717340	rice
...	..	..	...	...	...	...	...
107	34	32	26.774637	66.413269	6.780064	177.774507	coffee
99	15	27	27.417112	56.636362	6.086922	127.924610	coffee
118	33	30	24.131797	67.225123	6.362608	173.322839	coffee
117	32	34	26.272418	52.127394	6.758793	127.175293	coffee
104	18	30	23.603016	60.396475	6.779833	140.937041	coffee

Figura 16.. Extracto del set de datos Crop Recommendation.
Fuente: elaboración propia basada en la práctica.

3.2. Análisis del Dataset

Antes de iniciar el desarrollo del modelo, se realizó un análisis del conjunto de datos utilizado. El propósito principal de esta tarea fue comprender las características, la distribución y las relaciones entre los parámetros y sus respectivas etiquetas. Para lograrlo, se emplearon diversas herramientas, incluido ciertos gráficos. Además, se buscó identificar problemas de

completitud de los datos, es decir, analizar si todas las características estaban completas o existían datos faltantes. Luego, se llevó a cabo una búsqueda de valores atípicos (outliers) y de errores en los datos. Este proceso contribuyó a determinar la calidad general de los datos y el procesamiento necesario, antes de continuar con la etapa de entrenamiento.

3.2.1. Distribución de los datos

El primer paso del análisis del dataset consistió en comprender la distribución de las categorías en el conjunto, es decir, la cantidad de muestras que corresponde a cada categoría. Esto es fundamental, porque permite determinar si se está trabajando con un conjunto de datos balanceado o no. Para llevar a cabo esta tarea, se empleó la librería Pandas y se obtuvieron los resultados expresados en la figura 17.

```
rice           100
maize          100
jute           100
cotton         100
coconut        100
papaya         100
orange         100
apple          100
muskmelon      100
watermelon     100
grapes         100
mango          100
banana         100
pomegranate    100
lentil         100
blackgram      100
mungbean       100
mothbeans      100
pigeonpeas     100
kidneybeans    100
chickpea       100
coffee         100
```

Figura 17. Cantidad de muestras por categorías
Fuente: elaboración propia basada en la práctica.

Como se puede observar en la figura, el conjunto de datos se muestra completamente equilibrado, dado que se disponen de 100 muestras para cada clase. Es muy importante que el conjunto de datos esté balanceado, ya que un conjunto desbalanceado puede resultar en un rendimiento sesgado del modelo hacia las clases mayoritarias. Esto puede implicar en que el modelo tenga dificultades para generalizar cuando se trabaja con datos nuevos y, por consiguiente, presentar errores en la clasificación.

3.2.2. Completitud de los datos y detección de errores

A continuación, se llevó a cabo un análisis exploratorio para identificar la presencia de atributos incompletos, cuyos resultados se presentan en la figura 17. A partir de este

análisis se llegó a la conclusión de que todos los atributos contenían datos. Además, se procedió a evaluar si existían datos erróneos en el conjunto de datos, por lo tanto, se investigó el tipo de dato de cada atributo tal como se muestra en la figura 18. Al tratarse en su mayoría de atributos del tipo numéricos, se descarta la presencia de datos erróneos. Consecuentemente, el siguiente paso fue detectar valores atípicos (outliers). Cabe destacar que cuando se investigó el tipo de dato de cada atributo, se determinó que las categorías (llamadas "label" en el conjunto de datos) eran de tipo Object. Esta es una cuestión importante, ya que algunos algoritmos de clasificación requieren que las etiquetas sean del tipo numérico para poder procesarlas adecuadamente. Esto se debe a que estos algoritmos realizan operaciones matemáticas con las etiquetas, como cálculos de distancias, comparaciones y ajuste de parámetros, entre otros. En consecuencia, la conversión de las etiquetas a formato numérico es esencial para garantizar el correcto funcionamiento de los algoritmos.

```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
```

Figura 18. Completitud del conjunto de datos.
Fuente: elaboración propia, basada en la práctica.

```
N              int64
P              int64
K              int64
temperature  float64
humidity     float64
ph           float64
rainfall     float64
label         object
```

Figura 19. Tipo de datos de cada atributo
Fuente: elaboración propia basada en la práctica.

3.2.3. Detección de datos atípicos

Finalmente, se analizaron los datos atípicos[3] y en este caso se abordaron dos enfoques diferentes: el primero fue emplear un diagrama de cajas (Ver figura 20 a la 26) por cada atributo. Este tipo de diagramas brinda información sobre parámetros estadísticos tales como: la media, los cuartiles y la distribución de los datos respecto a ellos. Si la media es más próxima al primer cuartil (Q1), esto sugiere que predominan los valores bajos y que

[3] Datos atípicos: también conocidos como outliers, son datos que se encuentran por fuera de los límites superior o inferior, los cuales están definidos por el rango Intercuartil.

posiblemente existan algunos valores altos (atípicos) que estén ampliando la distribución en dirección al tercer cuartil (Q3). Por otro lado, si la media es más próxima al tercer cuartil, implica que predominan valores altos y que podrían existir valores atípicos hacia el primer cuartil. Un ejemplo de esta situación se puede visualizar en la Figura 20 para la categoría "apple" (manzana), donde la media se encuentra más cercana a Q1, en consecuencia, pueden existir outliers en los valores que comprenden al Q3. La misma situación se repite en la figura 23 para la categoría "Kidneybeans" (Frijoles Rojos), donde la media está más próxima al Q1, y, por lo tanto, pueden existir outliers en el rango Q3.

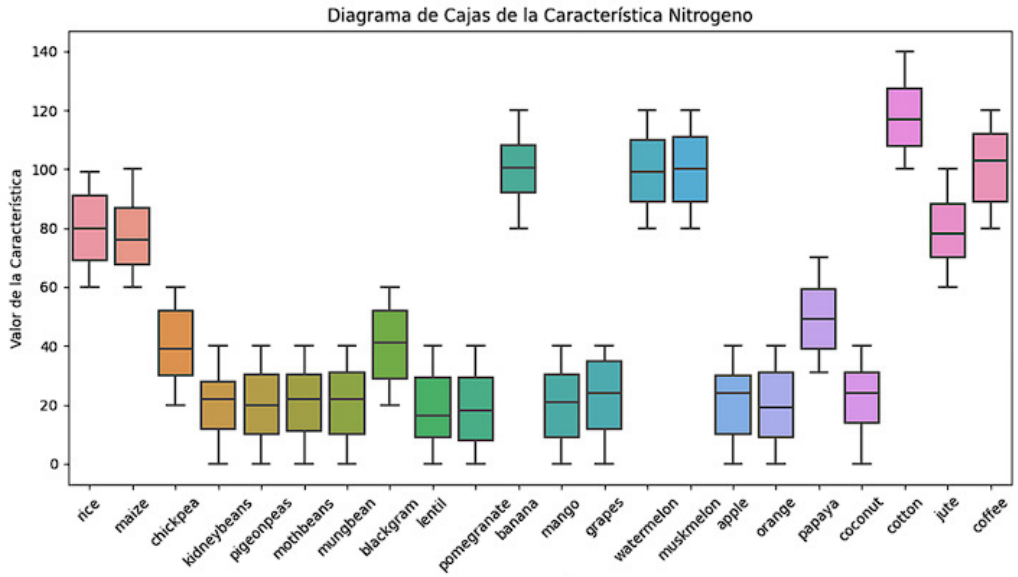


Figura 20. Diagrama de cajas para visualizar la distribución del Nitrógeno en cada una de las categorías.
Fuente: elaboración propia basada en la práctica.

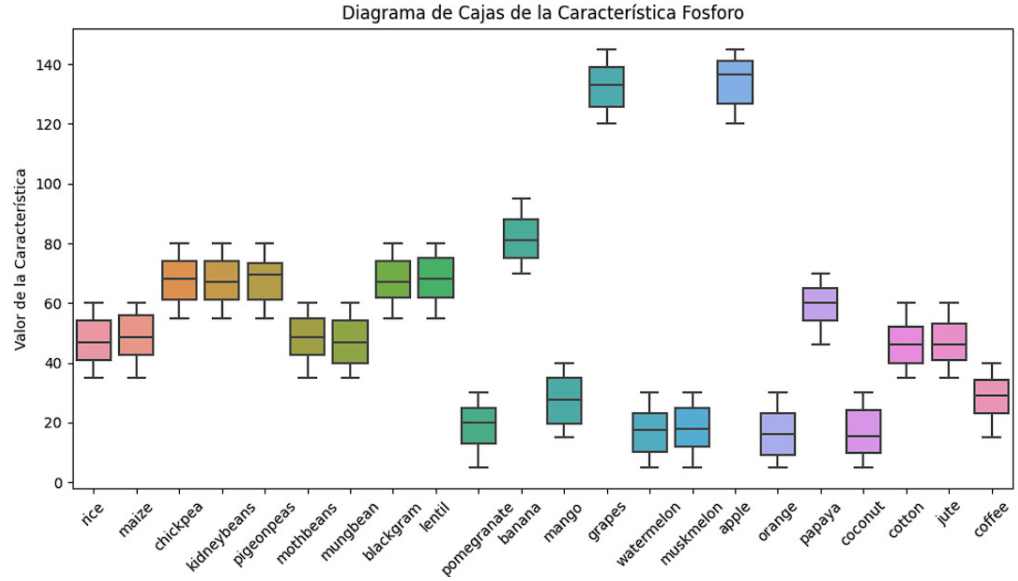


Figura 21. Diagrama de cajas para visualizar la distribución del Fósforo en cada una de las categorías.
Fuente: elaboración propia basada en la práctica.

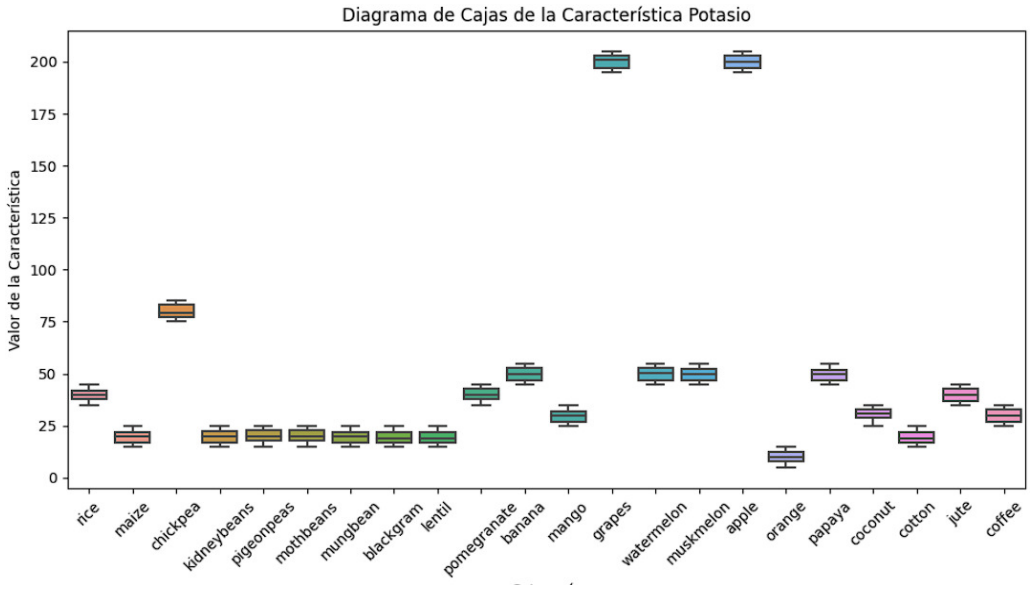


Figura 22. Diagrama de cajas para visualizar la distribución del Potasio en cada una de las categorías.
Fuente: elaboración propia basada en la práctica.

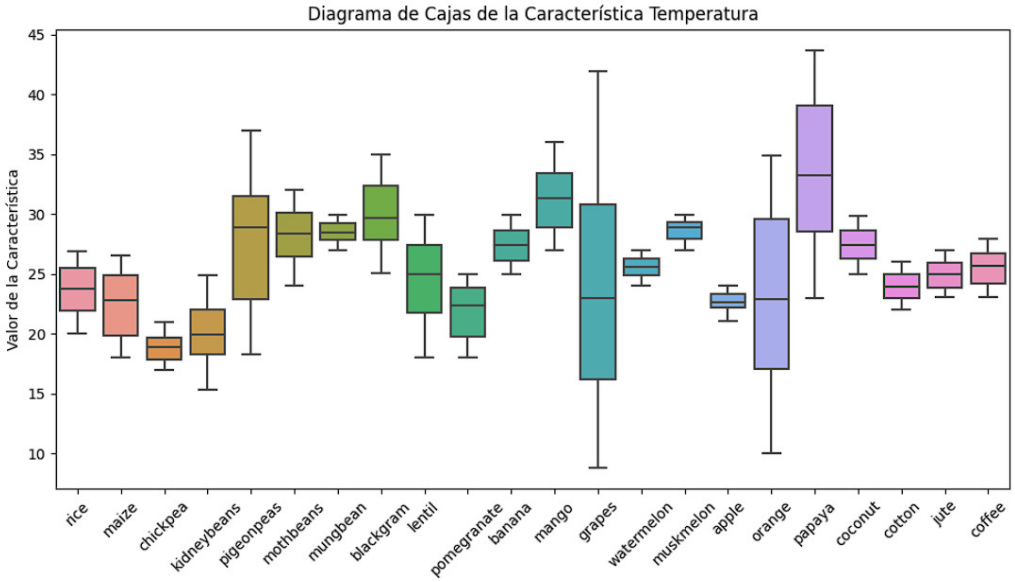


Figura 23. Diagrama de cajas para visualizar la distribución de la temperatura en cada una de las categorías.
Fuente: elaboración propia basada en la práctica.

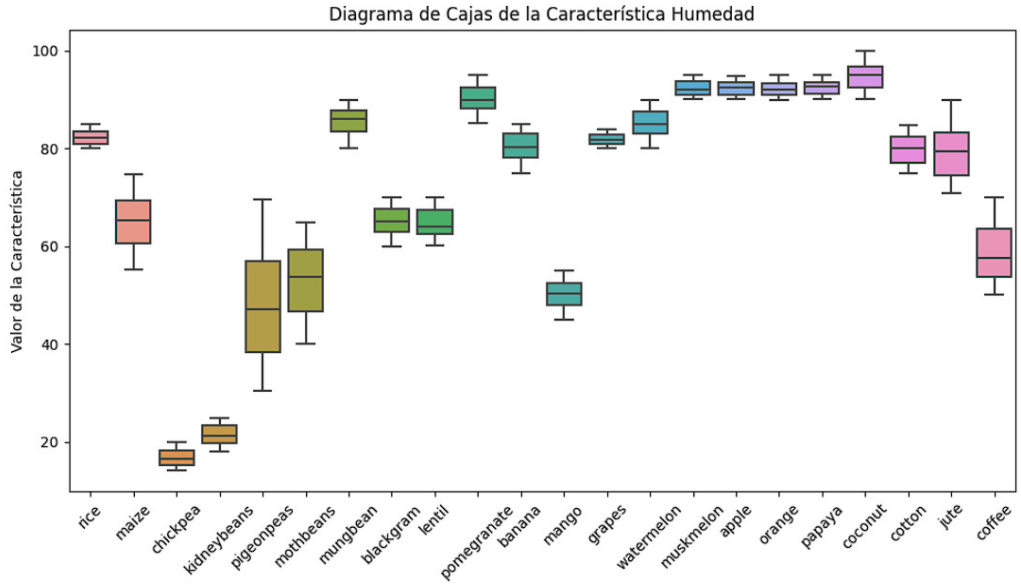


Figura 24. Diagrama de cajas para visualizar la distribución de la humedad relativa en cada una de las categorías.
Fuente: elaboración propia basada en la práctica.

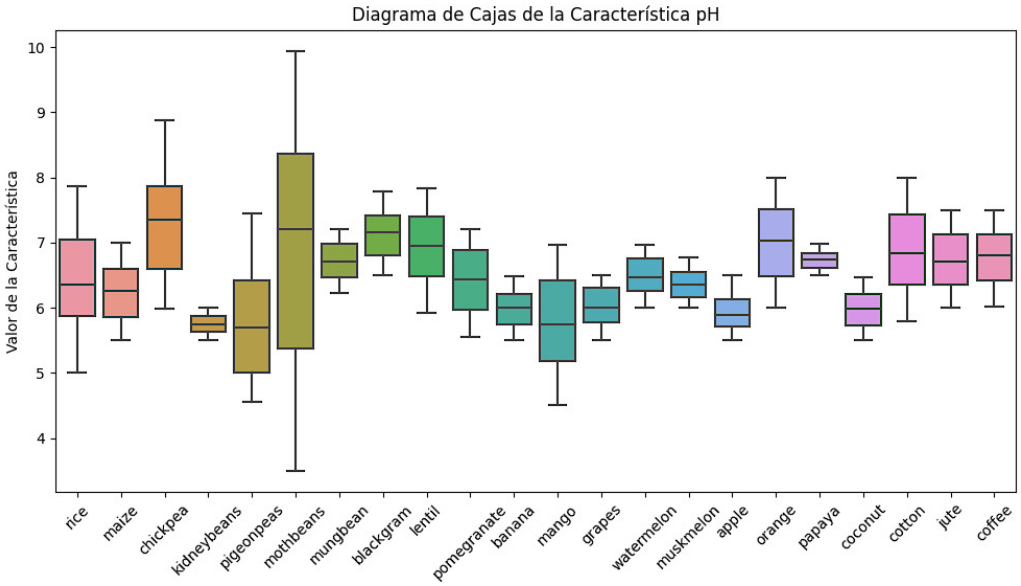


Figura 25. Diagrama de cajas para visualizar la distribución del pH en cada una de las categorías
Fuente: elaboración propia basada en la práctica.

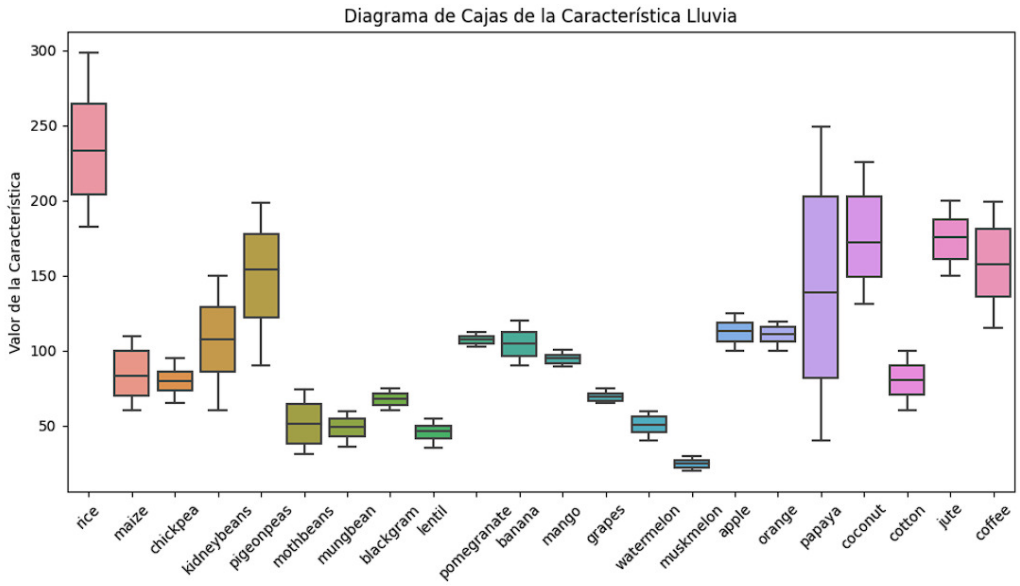


Figura 26. Diagrama de cajas para visualizar la distribución de la lluvia en cada una de las categorías.
Fuente: elaboración propia basada en la práctica.

Para descartar la presencia de outliers en el conjunto de datos analizado, se llevó a cabo el segundo enfoque, el cual es analítico. Este nuevo análisis consistió en el cálculo del Rango intercuartílico (IQR), definido como la diferencia entre el tercer cuartil y el primer cuartil. Luego, se definió un límite superior y otro inferior. De esta manera, cualquier valor que se encuentre por fuera de esos límites se lo denomina dato atípico. Los límites definidos se los pueden expresar de la siguiente forma:

- $límite\ inferior = Q1 - 1.5 * IQR$
- $límite\ superior = Q1 + 1.5 * IQR$

Por lo tanto, cualquier valor de q que cumpla con las siguientes condiciones se lo considera como un outlier leve:

$$q < Q1 - 1.5 * IQR, \text{ o bien } \ q > Q1 + 1.5 * IQR$$

El último proceso mencionado para el cálculo de outliers es iterativo, ya que se debe evaluar cada una de las características de cada una de las categorías del conjunto de datos.

Si bien pueden apreciarse algunas observaciones en el análisis de los datos por métodos gráficos, que sugieren la posible presencia de valores atípicos, el posterior cálculo analítico de los rangos intercuartiles descartó la existencia de tales valores atípicos en el conjunto de datos.

3.2.4. Preprocesamiento de los datos

Una vez finalizado el proceso de análisis del conjunto de datos, se obtuvo como conclusión que los mismos estaban balanceados, completos, que no existían errores en los

valores de los atributos y, además, no se detectaron outliers. En consecuencia, se determinó que el dataset es apto para entrenar un modelo de clasificación. Sin embargo, se modificaron los nombres de las etiquetas, para que las mismas sean descriptivas respecto al atributo que representa. Por otro lado, los datos numéricos fueron redondeados a dos cifras significativas, ya que en el set de datos original se emplearon seis cifras significativas. Este último cambio mencionado, se realizó para reducir la complejidad numérica, debido a que los cálculos se vuelven más simples y requieren menos recursos; además, este cambio permite que el algoritmo sea menos sensible a variaciones pequeñas y, por consiguiente, pueda converger más rápido.

3.2. Modelos de clasificación

En esta sección se abordarán los diferentes modelos de clasificación construidos a lo largo del desarrollo de la práctica profesional, incluyendo su entrenamiento y el rendimiento obtenido para cada uno de ellos.

Por otro lado, al momento de seleccionar los modelos a desarrollar se tuvo en cuenta los siguientes aspectos:

- Problemática que se desea resolver: el caso de estudio es un problema de clasificación por lo tanto se estudiaron los algoritmos que abordan este tipo de problemática.
- El tamaño del conjunto de datos con el que se va a trabajar: ya que si se trabaja con grandes conjuntos de datos es recomendable trabajar con redes neuronales, mientras que si trabaja con un menor número de muestras se puede trabajar con SVM.
- Composición de las características del conjunto de datos: este punto hace referencia a si las características son categóricas, numéricas o una combinación de ambas. Este punto es de suma importancia, debido a que algunos algoritmos no permiten variables categóricas en sus características, por lo tanto, es necesario realizar una conversión mediante alguna técnica, como one-hot. Otros algoritmos no permiten que las categorías sean nominales, por lo que nuevamente es necesario llevar a cabo una conversión.
- Velocidad de predicción: algoritmos como SVM, regresión logística y algunos tipos de redes neuronales son más rápidos al momento de realizar predicciones en comparación a otros, como K vecinos más cercanos (kNN), algoritmos de conjunto o redes neuronales profundas.
- Disponibilidad de memoria RAM para la carga del conjunto de datos: para el caso de estudio no existe una limitante en este aspecto ya que el set de datos es pequeño y

posee aproximadamente 2300 registros. Sin embargo, si se trabajara con un gran número de muestras y existiera un limitante respecto a la RAM es aconsejable trabajar con algoritmos de aprendizaje incremental, para que puedan mejorar el modelo en la medida que se agreguen datos gradualmente.

Una vez analizado cada uno de los aspectos anteriormente mencionados, se empleó como apoyo el diagrama de selección de algoritmos expuesto por Scikit-lern (ver figura 27) para determinar qué algoritmos analizar.

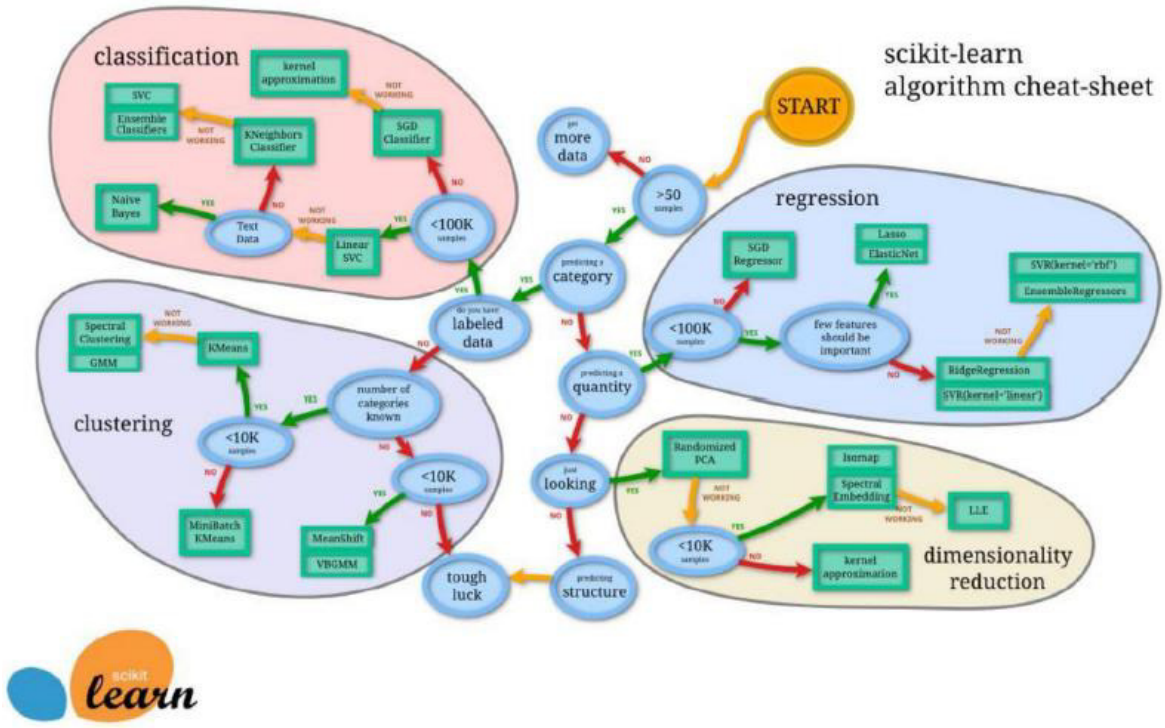


Figura 27. Diagrama de selección de algoritmos.
Fuente: Burkov, A (2019). “Basic Practice” en *The Hundred-Page Machine Learning Book*. Capítulo 5.2, p 9.

3.3.1. Support Vector Machine

El primer modelo que se desarrolló fue una Máquina de Vectores de Soporte (SVM: Support Vector Machine). Se utilizó la librería de scikit-learn para la construcción del mismo. Como se trabajó en un problema de clasificación multiclase se tuvo que hacer uso de una función Kernel. En primera instancia, se empleó una función de kernel Sigmoide, debido a que se realizó un análisis del gráfico de dispersión (ver figura 28) en la que se representan los atributos nitrógeno y lluvia, de cada una de las categorías de clasificación, que se encuentran señaladas en el lado derecho de la figura. Al disponer de una gran cantidad de categorías solapadas, se observó que no se podía establecer fácilmente un hiperplano de separación en la misma dimensión de los datos originales, por lo que se concluyó que los

datos no eran linealmente separables. Sin embargo, al realizar la validación cruzada con cross_val_scorel se obtuvo una tasa de aciertos de aproximadamente del 32%. Por lo tanto, se determinó que era necesario realizar un ajuste en los hiperparámetros del modelo, consecuentemente se utilizó la técnica GridSearchCV4 para determinar cuál de los siguientes tipos de kernel era el más adecuado: linear (Kernel lineal), poly (kernel polinómico), rbf (kernel que emplea la función de base radial) y, por último, sigmoid (kernel que emplea la función sigmoide). El resultado que se obtuvo fue que el kernel más óptimo era el lineal, el cual se suele emplear cuando los datos son aproximadamente linealmente separables, esto implica que no se requiere realizar una transformación de los datos a una dimensión superior.

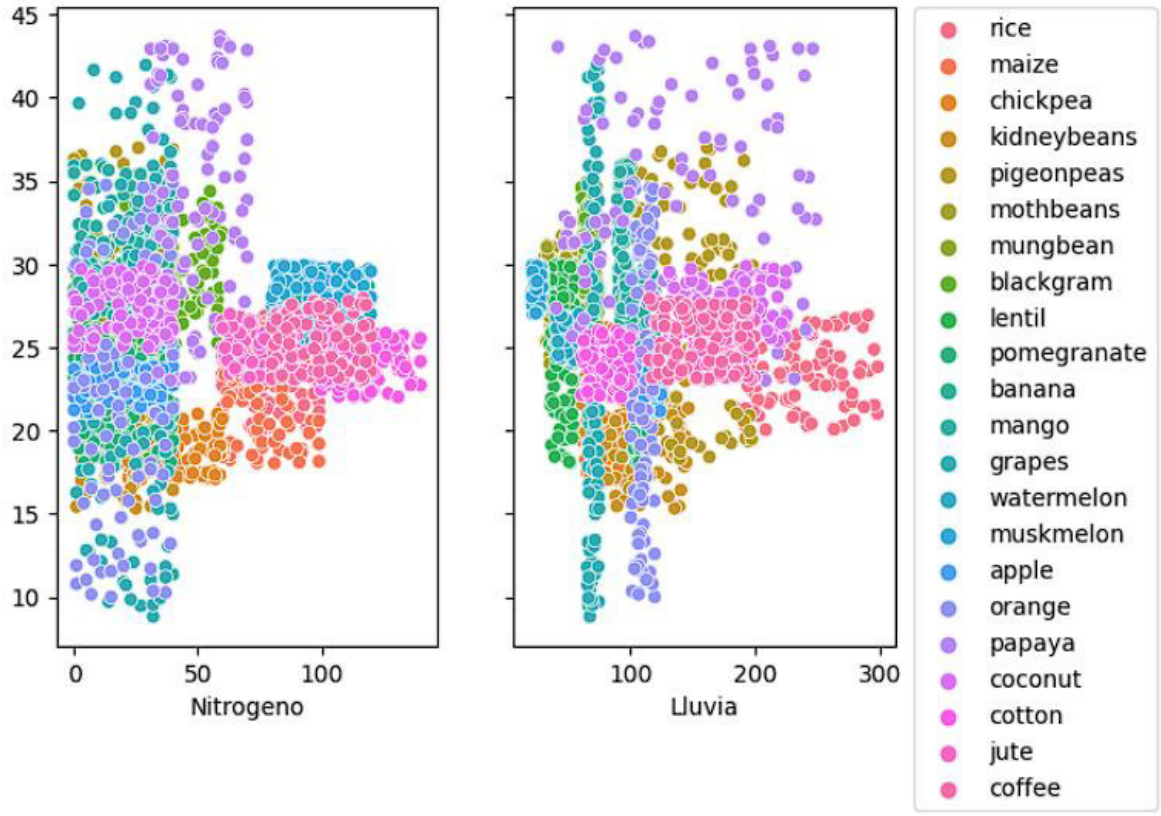


Figura 28. Gráfico de dispersión de los atributos Nitrógeno y Fósforo de cada una de las categorías de clasificación.
Fuente: elaboración propia basada en la práctica.

Por otro lado, también se utilizó la técnica GridSearchCV para determinar el valor de C[5] y se obtuvo como resultado el valor 5. Además, se estableció el decision_function_shape

[4] GridSearchCV: es una técnica que se emplea para obtener los hiperparámetros más óptimos del modelo con el que se está trabajando.
[5] C: es un parámetro de regularización que controla el equilibrio entre maximizar los márgenes entre las diferentes clases y minimizar los errores de clasificación.

[6]como 'ovr' [7], para que se cree un clasificador binario por cada categoría del conjunto de datos. Finalmente, se empleó la función de perdida Hinge y el optimizador SMO, que son las opciones por defecto de la librería.

Luego de ajustar los hiperparámetros, se llevó a cabo nuevamente la validación cruzada con cross_val_score y se obtuvieron los resultados que se pueden apreciar en la figura 29. El rendimiento promedio es del 98,6% en comparación al 32% que se había obtenido con la utilización del kernel de base radial.

```
Puntajes de cada pliegue: [0.977 0.984 0.989 0.989 0.991]
Puntaje promedio: 0.985909090909091
```

Figura 29. Rendimiento general del modelo y de cada pliegue.
Fuente: elaboración propia, basada en la práctica.

Como consecuencia de los resultados obtenidos de la validación cruzada, se concluyó que la selección de los hiperparámetros del modelo fue la correcta. Por lo tanto, se generó un nuevo modelo y, al momento de entrenarlo, se debió emplear todo el conjunto de datos, ya que esto lo dota de una mayor capacidad para generalizar. Sin embargo, al no contar con un conjunto de prueba independiente, para determinar que el mismo no se sobreajustó a las muestras, se debió separar el conjunto de datos en entrenamiento y pruebas. Por lo tanto, con el primero se entrenó el modelo con el método fit, y con el segundo se evaluó el rendimiento para determinar la capacidad de clasificar en datos no conocidos

3.3.2. Regresión Logística.

El segundo modelo que se desarrolló, empleó el algoritmo de Regresión Logística. Para optimizar la función de costos se aplicó el solver[8] newton-cg, aunque se podría haber elegido otro solver, tales como lbfgs (es el configurado por defecto), sag (que se lo suele emplear en conjuntos de datos grandes) y saga (que es una variante de sag); sin embargo, se comprobó que con newton-cg el modelo converge a una velocidad mayor, en comparación con los demás solver mencionados. Además, se observó que el algoritmo requería de al menos 266 iteraciones para lograr la convergencia. Por otro lado, se empleó el regularizador L2 y la función de costos multinomial, que son las configuradas por defecto en la librería.

[6] Decision_function_shape: es un parámetro que se debe definir en los modelos de clasificación multi-clase, para indicar qué estrategia se empleará para resolver la clasificación.
[7] Ovr: cuando se indica como estrategia ovr, se crea un clasificador binario por cada categoría y se contemplan las dos posibilidades, positivo o negativo.
[8] Solver: en el contexto de un modelo que emplee la regresión logística como algoritmo, los solver son optimizadores específicos utilizados para encontrar los coeficientes óptimos del modelo.

Una vez definidos los ajustes de los hiperparámetros se llevó a cabo la validación cruzada con cross_val_score que es una de las técnicas más sencillas de emplear. Además, permite estimar la capacidad de generalización del modelo y, posteriormente, realizar los ajustes necesarios. Es importante resaltar, que cross_val_score tiene la finalidad de estimar el rendimiento, pero no retorna un modelo capaz de clasificar datos nuevos, por lo tanto, sirve como guía para conocer la efectividad del modelo para resolver el problema de clasificación.

En la figura 30 se puede apreciar un fragmento de código en el cual se genera una instancia cross_val_score y se pasaron como parámetros: el clasificador, que en este caso es el de regresión logística, la “x” representan las variables independientes, que son los atributos de todo el conjunto de datos de entrenamiento y la “y” representa la variable dependiente, que son las categorías de todo el conjunto de datos, por último, se definió la métrica con la se evaluó el rendimiento general del modelo que es f1-score.

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score, make_scorer
f1_scorer = make_scorer(f1_score, average='macro')
scores = cross_val_score(clasificador, x, y, cv=5, scoring=f1_scorer )
print("Puntajes de cada pliegue:", scores)
print("Puntaje promedio:", scores.mean())
```

Figura 30. Validación cruzada con cross_val_score.
Fuente: elaboración propia basada en la práctica.

Los resultados que se obtuvieron de cada pliegue o subconjunto con cross_val_score, son los siguientes:

- Primer pliegue 98,9%
- Segundo pliegue 98,6%
- Tercero pliegue 98,6%
- Cuarto Pliegue 98 %
- Quinto pliegue 98.4%

El resultado del rendimiento promedio general obtenido teniendo en cuenta la métrica F1 Score fue de 98.5% de precisión.

Como método adicional de validación cruzada se empleó la clase StratifiedKFold[9] en el conjunto de entrenamiento, para realizar una división de k pliegues con la que se entrenó y validó el modelo. Se eligió esta clase, debido a que asegura que cada subconjunto o pliegue contenga la misma cantidad de muestras por categoría. Luego se llevó a cabo el proceso de

[9] StratifiedKFold es una clase de la librería scikit-learn empleada para realizar validación cruzada.

validación de cruzada (ver Figura 31). En ella se puede visualizar que el conjunto de datos fue subdividido en 5 subconjuntos o pliegues. Cada uno de ellos debía tener aproximadamente el mismo tamaño, y las categorías debían estar mezcladas de tal forma que cada pliegue se encuentre balanceado. Estos 5 pliegues fueron recorridos mediante el uso de un bucle for y, en cada iteración el modelo fue entrenado y validado. De cada pliegue se pudo conocer el rendimiento del modelo en el entrenamiento (ver figura 32).

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

def cv(x, y ):
  fold_no = 1
  accuracy_scores = []
  for train_index, test_index in skf.split(x, y):
      X_train, X_test = x.iloc[train_index], x.iloc[test_index]
      y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

      clasificador.fit(X_train, y_train_fold)
      y_pred = clasificador.predict(X_test)
      accuracy = accuracy_score(y_test_fold, y_pred)
      print('Fold', str(fold_no), 'Accuracy:', accuracy)
      fold_no += 1
      accuracy_scores.append(accuracy)
  return accuracy_scores

accuracy_scores = cv(x, y)
y_pred_test = clasificador.predict(x)
average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
print("Promedio del clasificador", average_accuracy )
print(classification_report(y, y_pred_test))
```

Figura 31. Aplicación de validación cruzada con StratifiedKFold.
Fuente: elaboración propia basada en la práctica.

```
Fold 1 Accuracy: 0.9801136363636364
Fold 2 Accuracy: 0.9886363636363636
Fold 3 Accuracy: 0.9744318181818182
Fold 4 Accuracy: 0.9857954545454546
Fold 5 Accuracy: 0.9829545454545454
```

Figura 32. Accuracy de cada pliegue con StratifiedKFold.
Fuente: elaboración propia basada en la práctica.

Con los resultados obtenidos de ambos métodos de validación cruzada se pudo concluir que los hiperparámetros seleccionados son los adecuados, ya que en promedio se logró una precisión cercana al 98%. Por lo tanto, se generó un nuevo modelo y, al igual que en el caso de entrenamiento del modelo que emplea SVM, se debió dividir el conjunto de datos en dos. El primero, está destinado al entrenamiento del modelo, y representa el 80%

del total de las muestras, mientras que el segundo conjunto es el de prueba, con este se evaluó la capacidad del modelo de clasificar datos nuevos.

3.3.3. Red Neuronal Densa.

Para este caso, se empleó la librería de Tensor Flow para el desarrollo de una red neuronal Secuencial y Densa. Fue necesario realizar una transformación en los datos, ya que los mismos eran Dataframe10, debido a que se había empleado la librería pandas para leer el archivo .csv, que contenía el conjunto de datos con el que se trabajó. Sin embargo, un modelo desarrollado con Tensor Flow, requiere que los datos sean del tipo tensores11. Es por ello que primero, se dividió el conjunto de datos en un conjunto de entrenamiento, que representaron aproximadamente el 80% del total y, el restante fue el conjunto de prueba. Posteriormente, se procedió a transformar tanto el conjunto de prueba como el de entrenamiento en tensores.

Luego, se debió reemplazar las diferentes categorías que eran string a enteros. Para llevar a cabo esta conversión existen dos enfoques. El primero consiste en reemplazar cada una de las categorías por números. Este enfoque se lo suele abordar cuando se trabaja con variables ordinales12, mientras que si se trabaja con variables nominales13 se emplea el enfoque one-hot. Esto consiste en añadir una nueva variable binaria por cada categoría y en las observaciones, que pertenezcan a esa categoría contendrán un valor de 1 en caso de ser positivo, de lo contrario un 0.

A pesar de que las categorías que conforman el set de datos son variables nominales, se consideró que era más apropiado emplear el primer enfoque, ya que, al ser 22 categorías, utilizar el método one-hot implicaría aumentar significativamente la dimensionalidad de los datos.

Posteriormente, se tuvo que definir los hiperparámetros del modelo, para ello se podía seguir dos enfoques, el primero, era probar de forma manual el número de capas y la cantidad de neuronas que debían contener, luego evaluar el rendimiento y realizar los ajustes que fueran necesarios. Este proceso puede ser muy largo y poco eficiente, por lo tanto, se empleó

[10] Dataframe: es una estructura de datos de tamaño variable, tabulares y bidimensionales. Los datos se estructuran en filas y columnas.
[11] Tensores: son matrices multidimensionales, con un tipo uniforme de datos. Son inmutables, es decir, que no pueden ser modificados.
[12] Variables ordinales: son palabras, pero se les puede dar un orden jerárquico, es decir, que se puede establecer un orden de precedencia.
[13] Variables nominales: son palabras, pero no se les puede dar ningún tipo de orden de procedencia a las categorías y un ejemplo de ello, son categorías de frutas.

el segundo enfoque, que consiste en la búsqueda de los hiperparámetros por medio de funciones expuestas por librerías. Como el modelo se lo iba a crear empleando Tensor-Flow, se utilizó la función RandomSearch de la librería kerastuner. Finalmente, el resultado que se obtuvo fue que la red debía estar compuesta por 3 capas internas. En la capa de entrada, el número de neuronas se determina por la cantidad de atributos en las muestras, que en este caso fue 7. Luego, la primera capa interna se compuso por 128 neuronas, y las siguientes se contuvieron 256 neuronas cada una. Por último, la capa de salida en los problemas de clasificación debe tener el mismo número de neuronas que categorías en el conjunto de datos, por lo tanto, se definió 22 neuronas. Además, se empleó la función de activación ReLU en las capas ocultas, mientras que la última capa utilizó Softmax, especialmente diseñada para problemas de clasificación multiclase.

Es importante resaltar que se definió la función de pérdida SparseCategoricalCrossentropy, que se la emplea cuando se está trabajando con más de una categoría y las mismas están representadas por números enteros. Además, se empleó el optimizador Adam y se definió que el modelo debía recorrer 70 veces el conjunto de datos de entrenamiento.

Al igual que en los modelos anteriormente expuestos, se llevó a cabo la validación cruzada y se midió empleando la métrica Accuracy esto nos permite conocer el rendimiento del modelo en diferente conjunto de entrenamiento y validación. Los resultados que se obtuvieron son los expuestos en la imagen 33. En ella se puede visualizar que en general el modelo alcanzó una precisión cercana al 96%, por lo tanto, los hiperparámetros seleccionados son acordes al conjunto de datos.

```
11/11 [==============================] - 0s 2ms/step - loss: 0.0830 - accuracy: 0.9773
11/11 [==============================] - 0s 2ms/step - loss: 0.0770 - accuracy: 0.9716
11/11 [==============================] - 0s 3ms/step - loss: 0.0823 - accuracy: 0.9602
11/11 [==============================] - 0s 3ms/step - loss: 0.1452 - accuracy: 0.9375
11/11 [==============================] - 0s 3ms/step - loss: 0.1737 - accuracy: 0.9403
0.9573863744735718
```

Figura 33. Indica el rendimiento general del modelo y de cada uno de los subconjuntos.
Fuente: elaboración propia, basada en la práctica.

Con los hiperparámetros definidos se llevó a cabo el entrenamiento del modelo y, en la figura 34, se puede ver la evolución de la precisión conforme se ejecutan las épocas.

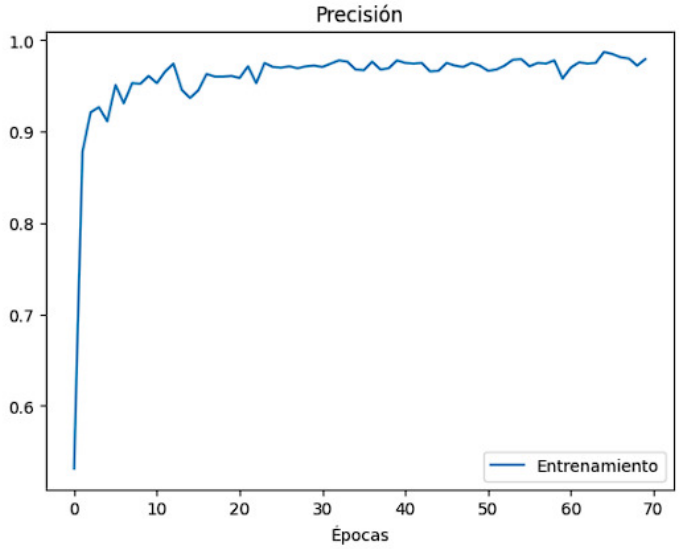


Figura 34. Evolución de la precisión en el conjunto de entrenamiento.
Fuente: elaboración propia basada en la práctica.

Tras analizar los 3 modelos construidos, se determinó que el último expuesto es el que se debía emplear en la aplicación desarrollada en Android. Esta decisión se basó en la compatibilidad entre las librerías Scikit-learn, Tensor Flow y Android. Para llevar a cabo la integración entre el modelo de aprendizaje automático y Android, es necesario que éste sea del tipo TensorFlow Lite y, debido a que no existe una compatibilidad directa Android y otras librerías, para este tipo de integraciones, se optó por el modelo desarrollado en TensorFlow. Es importante aclarar, que se podría haber abordado un enfoque que consiste en convertir los modelos construidos con Scikit-learn a TensorFlow. Sin embargo, esta conversión no es totalmente segura debido a que ambas librerías no son intrínsecamente compatibles, ya que presentan diferencias en las arquitecturas de cada una.

4. Desarrollo de la aplicación Móvil

Para el desarrollo de la aplicación móvil se utilizó el lenguaje de programación Kotlin y el IDE Android Studio. Un IDE es un entorno de desarrollo integrado que no solo permite escribir y ejecutar código, sino que proporciona un conjunto de herramientas que hace que el proceso de desarrollo sea más simple y eficiente. Algunas de las herramientas que proporciona Android Studio son las siguientes: compilación y depuración de código, sintaxis inteligente, emulación de dispositivos móviles, entorno unificado para el desarrollo de todo tipo de dispositivos Android, integración con GitHub, herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones, entre otras.

Las especificaciones del equipo para la instalación del IDE son los detallados en la figura 35.

Requisito	Mínimo	Recomendado
SO	Microsoft Windows 8 de 64 bits	La versión más reciente de Windows de 64 bits
RAM	8 GB de RAM	16 GB de RAM o más
CPU	Arquitectura de la CPU de x86_64; procesador Intel Core de segunda generación o posterior, o CPU AMD compatible con un framework de hipervisor de Windows	Procesador Intel Core más reciente
Espacio en el disco	8 GB (IDE y SDK de Android y Emulator)	Unidad de estado sólido con 16 GB o más
Resolución de pantalla	1280 x 800	1920 x 1080

Figura 35. Especificaciones que debe tener un equipo para poder instalar Android Studio.
Fuente: Recuperado de https://developer.android.com/studio/install?hl=es-419

Por otro lado, para llevar a cabo las diferentes funcionalidades de la aplicación se hizo uso de las siguientes librerías:

- Dagger hilt: para llevar a cabo la inyección de dependencia.
- MPAndroidChart y DonutGaugeView: se empleó esta librería para la representación gráfica de la información.
- Picasso: es una librería que permitió la carga de imágenes mediante url en un componente del tipo imageView.
- TensorFlow-lite: se lo empleó para la gestión del modelo dentro de la aplicación Android.
- Lottie: se encargó de gestionar las animaciones que se emplean cuando está en proceso de carga de la información, previamente a ser presentar la información al usuario.

Además, se emplearon dependencias de Android que están orientadas a componentes optimizados para ciclos de vida de las actividades y fragmentos y, para la integración entre la aplicación Mobile y el modelo de aprendizaje, se empleó el servicio de Firebase[14], Machine Learning.

También se emplearon otros servicios Firebase, los cuales son: Firestore Database: que es una base de datos Cloud, no relacional y altamente flexible y escalable y, Authentication que proporciona diversos proveedores para el proceso de autenticación de un usuario con una aplicación, ya sea web o móvil.

[14] Firebase es una plataforma Cloud, que ofrece una amplia gama de herramientas y servicios, algunos de los cuales son gratuitos y que sirven para el desarrollo de las aplicaciones web y móviles

4.1. Integración del modelo con la aplicación

Luego de convertir el modelo de red neuronal densa de TensorFlow a TensorFlow Lite, se realizó la descarga de la misma. Para integrar dicho modelo, con Android existen diferentes enfoques, para el desarrollo de esta Práctica Profesional Supervisada se hizo uso de la plataforma Firebase.

Para integrar Android con Firebase, en primer lugar, fue necesario registrar la aplicación y para ello se debe indicar el nombre de la misma, que se encuentra en el archivo Gradle, de la aplicación Android. En la plataforma, luego y de forma opcional, se puede registrar el SHA-1. Posteriormente, se debe añadir el archivo de configuración de Firebase al proyecto Android.

Una vez concluidos los pasos anteriores, en la consola de Firebase, se debe seleccionar la opción “Machine Learning” y cargar el archivo. tflite, del modelo.

En cuanto a la aplicación, se desarrolló una clase llamada MachineLearning, en ella se implementó un método que facilita la conexión a Firebase para la descarga y almacenamiento local del modelo. Se adoptó este enfoque para evitar descargas innecesarias del modelo, optimizando la eficiencia de la aplicación. Además de gestionar la descarga del modelo, este método setea el Interpreter, que es una interfaz para la ejecución del modelo y es el encargado de realizar las inferencias.

La clase MachineLearning también incluye otros métodos esenciales, que son los siguientes:

- Creación del tensor de entrada: se construye en función de la información de la huerta asociada a un usuario.
- Creación del tensor de salida: en él se almacenan los resultados de la inferencia.
- Inferencia del modelo: este recibe como parámetros los tensores de entrada y de salida y una vez concluida la inferencia, los resultados son almacenados en los tensores de salida.
- Interpretación de los resultados: este método recibe como parámetros una lista con todas las posibles etiquetas de clasificación y, mediante el uso del tensor de salida (que ya contiene el resultado de la inferencia), se puede conocer el índice y determinar en la lista de etiquetas cuál fue la predicha, además de conocer la probabilidad de acierto.

4.2. Requerimientos de la aplicación

Para obtener el modelo que se encarga de realizar la clasificación de los diferentes cultivos es necesario que la aplicación cuente con internet, por ello se debió agregar en el archivo manifest el permiso de acceso a internet. Como este no implica un riesgo de seguridad para el dispositivo no es necesario que el usuario lo autorice. Además, como la aplicación solamente emplea como DataSource la base de datos Cloud Firestore y no existe ningún tipo de almacenamiento local, es necesario que para el funcionamiento correcto de la misma el dispositivo siempre se encuentre con conectividad a internet.

Por otro lado, es importante resaltar que la versión mínima de Android permitida es API 24, que abarca a los dispositivos con Android 7 y superiores; sin embargo, el target SDK o nivel de API de destino es para los dispositivos con Android 13. Esto último, se determinó debido a las políticas de seguridad publicadas por parte de PlayStore, que obliga a que todas las aplicaciones ya sean nuevas o publicadas. Además, tiene que tener como objetivo Android API 33, debido a que se implementaron una serie de mejoras en cuanto seguridad, privacidad y rendimiento. En síntesis, la aplicación puede funcionar correctamente en aquellos dispositivos que posean de Android 7 al 13.

4.3. Diseño de la UI de la aplicación.

La Aplicación se encuentra dividida en la capa de interfaz de usuario que está conformada por dos actividades y seis fragment[15]. Además, se creó una clase abstracta y en ella se definieron una serie de métodos, que son comunes en todos los fragmentos, como inflar las vistas, definir el binding, dar visibilidad a las animaciones, detenerlas y la lógica para mostrar mensajes al usuario. Por otro lado, se establecieron otros métodos abstractos cuya implementación depende del fragmento que extiende de la clase, algunos de ellos se encargan de setear los componentes necesarios de la UI u observar los eventos a los que están suscriptos. Es importante aclarar, que los 6 fragmentos extienden de esta clase base.

Dado que los fragment no realizan llamadas asincrónicas, debido a que no es una práctica recomendable, ya que pueden bloquear el hilo principal de la aplicación, además, de que viola el principio de responsabilidad única, se definió una clase ViewModel por cada fragment. Estos son los encargados de controlar los eventos y delegarlos a las otras capas

[15] Fragment están conformados por layout, es ellos se definen los componentes de la UI con los que interactúa el usuario, como botones, textview, edittext, entre otros. En estos casos los layout están enlazados a una clase que extiende de Fragment, y es en estos últimos donde se controlan los eventos realizados por el usuario

de la jerarquía. Por lo tanto, son los encargados de realizar las llamadas asincrónicas a la capa de dominio, que a su vez llama a la capa de datos y esta se conecta con el DataStore para obtener los datos. Para poder llevar a cabo esta tarea, se emplearon corrutinas y se definido que se debían ejecutar en el Scope del ViewModel. Una vez que se obtuvo una respuesta, esta es almacenada en un contenedor de datos observable, en particular se emplearon MutableLiveData, pero estos se definieron como privados dentro de la clase ViewModel. Para que los fragmentos se puedan suscribir a estos y recibir la información obtenida del DataStore, se emplearon LiveData, y tienen la particularidad que no pueden ser modificados, de esta forma se garantizó que los cambios en los valores solo se puedan realizar dentro de la clase ViewModel.

En el método onCreate de la clase abstracta se estableció el método que observa a estos LiveData y, ante cualquier cambio de estado en ellos, notifica a todos los observadores (fragmentos), que posean un de ciclo de vida activo, para que se lleven a cabo las operaciones necesarias en el UI.

4.4. Diseño de la capa de datos

La capa de datos en la aplicación emplea exclusivamente la base de datos Cloud Firestore como fuente principal de datos (DataSource) y para persistencia de los mismos. En esta base de datos, se definieron dos colecciones principales:

- Crop: que contiene información general de los cultivos y en ella se distinguen cuatro documentos:
    - CropInfo: contiene una breve descripción y una URL representativa para cada cultivo. La información mostrada en la pantalla "Cultivo Seleccionado" se extrae de esta colección.
    - LabelClasfication: este documento es una colección que contiene las etiquetas que el modelo puede predecir. Si se cambia el modelo, por otro que pueda clasificar un número mayor de etiquetas, este documento debe ser actualizado.
    - ValuesMAxMin: es una colección de documentos que tiene los valores máximos y mínimos de lluvia de cada cultivo.
    - ListCrop: está conformado por una colección de documentos, donde cada uno de ellos tiene una información aproximada del pH, temperatura, humedad, potasio nitrógeno y fósforo, que son necesarios en el suelo para que un cultivo

pueda desarrollarse. La información contenida es la que se visualiza en la pantalla “Tipos de cultivos”

- UserLandData: está conformado por un único documento, el cual está relacionado a los usuarios que se loguean en la aplicación. Este documento está conformado por una colección de documentos, donde cada uno de ellos contiene información de los parámetros del suelo de la huerta asociada al usuario y representa el histórico de la huerta.

En la aplicación, cada uno de estos documentos y colecciones fueron modelados en data class y, para poder lograrlo, se definió una clase Singleton que contiene funciones de extensión. Además, tienen por objetivo mapear la respuesta de FireStore, y son del tipo MutableMap<String, Any> en dtaclass diferentes. Consecuentemente se pudo representar la estructura de los datos de una forma más simple, manejable y que solo contemple los datos que son relevantes.

Por otro lado, se desarrollaron por cada data class su respectiva clase repository, que representa el único punto de acceso a la información y, por ende, son las clases que contienen las consultas que son relevantes en la aplicación. Es importante resaltar, que las operaciones de eliminar, actualizar o insertar datos concernientes a los cultivos, no son casos de uso definidos, por lo tanto, dicha funcionalidad no existe en la aplicación.

Es relevante señalar, en la base de datos se llevó a cabo la definición de una regla de seguridad, que permite que se puedan consultar los datos contenidos en los documentos aquellos usuarios que estén logueados dentro de la aplicación. Esta medida evita que la base de datos sea pública y que cualquier persona pueda acceder a los datos contenidos en ella.

4.5. Diseño de la capa de dominio

En esta capa se definieron los casos de uso que contiene la lógica de la aplicación y métodos, que son requeridos por múltiples clases y el objetivo es evitar la duplicidad de código. Consecuentemente, se llevó a cabo el desarrollo de dos clases, una de ellas está orientada a los pasos necesarios para la creación y ejecución del modelo, esta emplea los métodos de la clase MachineLearnig expuestos anteriormente en el apartado de “Integración del modelo”. Una vez que se construyó y ejecutó el modelo, como resultado de la aserción se obtiene un número que corresponde a la categoría clasificada. Luego, con este se consulta a la clase Repository relacionada con el documento "CropInfo,", tal como se expuso en la sección "Diseño de la capa de datos", y se obtiene información concerniente al cultivo como su nombre y otros datos que pueden ser de interés. Finalmente, el ViewModel relacionada a

la pantalla Cultivo seleccionado, obtiene el resultado y notifica a la UI que ocurrió un cambio de estado y que debe presentar al usuario los datos obtenidos

Por otro lado, en la segunda clase mencionada, se definieron una serie de métodos que tienen como objetivo ayudar a la UI en la representación de la información. Estos métodos se encargan de realizar los cálculos, que son necesarios para representar en un formato adecuado las fechas en el gráfico y que se visualiza en la pantalla Histórico de la huerta. Además, facilita la representación de los datos de la lluvia pertenecientes a los cultivos de una manera comprensible para el usuario. Esta funcionalidad es empleada, tanto en la pantalla Cultivo Clasificado, como en Detalles del cultivo. Finalmente, también se definieron métodos que son empleados en la pantalla Histórico de la huerta, estos ayudan a poder representar los datos de humedad y lluvia en el gráfico.

5. Resultados

En esta sección se muestran los resultados obtenidos, tanto de las métricas de los modelos desarrollados, como del funcionamiento de la aplicación.

5.1. Resultados de los modelos Desarrollados

5.1.1. Regresión Logística

Una vez entrenado y validado el modelo, se comprobó el rendimiento del mismo y para ello se emplearon datos prueba. Estos no se utilizaron previamente, por lo tanto, para el modelo eran nuevos y los resultados obtenidos de la clasificación se encuentran expresados en la figura 36.

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	20
banana	1.00	1.00	1.00	20
blackgram	1.00	1.00	1.00	20
chickpea	1.00	1.00	1.00	20
coconut	1.00	1.00	1.00	20
coffee	0.95	1.00	0.98	20
cotton	1.00	1.00	1.00	20
grapes	1.00	1.00	1.00	20
jute	0.86	0.90	0.88	20
kidneybeans	1.00	0.95	0.97	20
lentil	1.00	0.90	0.95	20
maize	1.00	1.00	1.00	20
mango	1.00	1.00	1.00	20
mothbeans	0.87	1.00	0.93	20
mungbean	1.00	1.00	1.00	20
muskmelon	1.00	1.00	1.00	20
orange	1.00	1.00	1.00	20
papaya	1.00	0.95	0.97	20
pigeonpeas	1.00	1.00	1.00	20
pomegranate	1.00	1.00	1.00	20
rice	0.89	0.85	0.87	20
watermelon	1.00	1.00	1.00	20
accuracy			0.98	440
macro avg	0.98	0.98	0.98	440
weighted avg	0.98	0.98	0.98	440

Figura 36. Métricas de los resultados de clasificación de cada categoría (tipo de cultivo) empleando un modelo cuyo algoritmo es el de regresión logística.
Fuente: elaboración propia basada en la práctica.

Las métricas generales del modelo fueron las mismas, tanto para la métrica Accuracy, como para la medición de F1-score empleando las técnicas weighted y Macro. Esto último, se debe a que se tiene una distribución balanceada de cada clase.

Por otro lado, se empleó la matriz de confusión (ver Figura 37) para visualizar la cantidad de muestras que fueron clasificadas correctamente, así como aquellas en las que el modelo cometió errores. La diagonal principal corresponde a las clasificaciones correctas, mientras que los valores restantes indican cuántas muestras se clasificaron de forma errónea en otra categoría. Por ejemplo, de las 20 muestras que pertenecían a la clase de arroz, 17 fueron clasificadas correctamente, mientras que las 3 restantes fueron catalogadas erróneamente como pertenecientes a la categoría de jute.

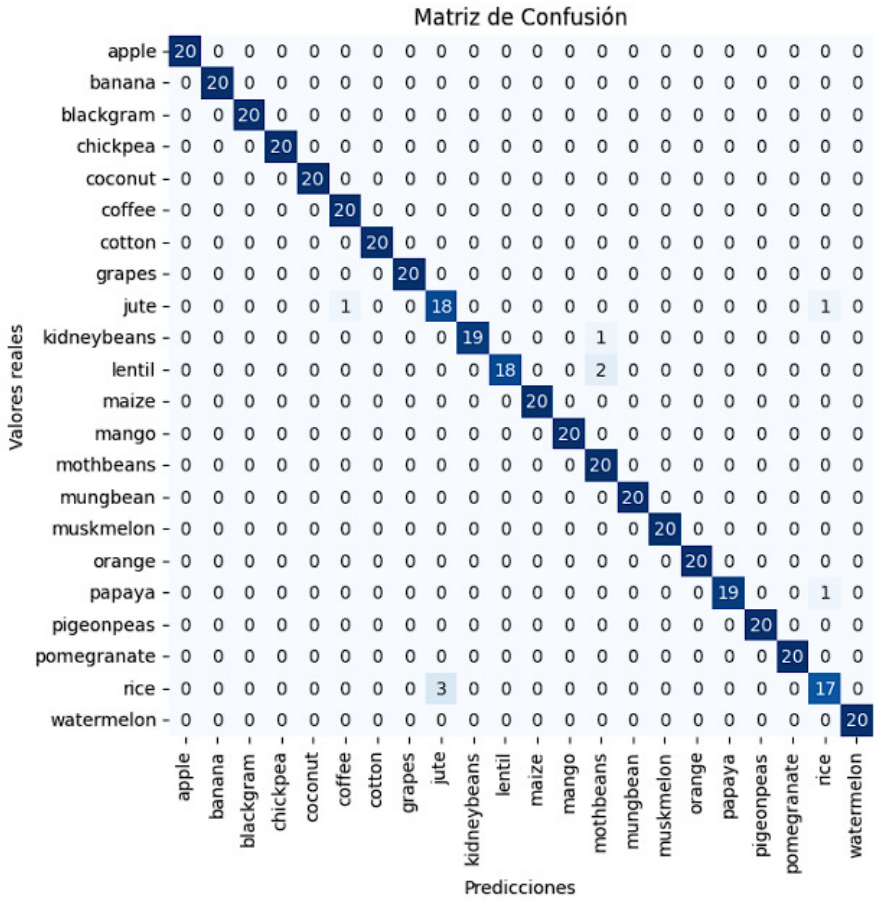


Figura 37. Matriz de confusión del modelo que emplea el algoritmo de regresión logística.
Fuente: elaboración propia basada en la práctica.

5.1.2. Support Vector Machine

Se midió el desempeño general del modelo con los datos de prueba y el resultado que se obtuvo es el representado en la figura 38. La columna support indica la cantidad de muestras de cada categoría y se puede apreciar que los datos de prueba se encuentran balanceados. Se midió el rendimiento del modelo al clasificar los datos en cada categoría con las métricas precisión, recall y f1-score. Finalmente, en la margen inferior de la figura, se pueden visualizar las métricas precisión (Accuracy) y f1-score (empleando técnicas macro y weight) general del modelo que, en ambos casos fue 99%.

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	20
banana	1.00	1.00	1.00	20
blackgram	1.00	1.00	1.00	20
chickpea	1.00	1.00	1.00	20
coconut	1.00	1.00	1.00	20
coffee	1.00	1.00	1.00	20
cotton	1.00	1.00	1.00	20
grapes	1.00	1.00	1.00	20
jute	0.87	1.00	0.93	20
kidneybeans	1.00	1.00	1.00	20
lentil	1.00	0.95	0.97	20
maize	1.00	1.00	1.00	20
mango	1.00	1.00	1.00	20
mothbeans	0.95	1.00	0.98	20
mungbean	1.00	1.00	1.00	20
muskmelon	1.00	1.00	1.00	20
orange	1.00	1.00	1.00	20
papaya	1.00	1.00	1.00	20
pigeonpeas	1.00	1.00	1.00	20
pomegranate	1.00	1.00	1.00	20
rice	1.00	0.85	0.92	20
watermelon	1.00	1.00	1.00	20
accuracy			0.99	440
macro avg	0.99	0.99	0.99	440
weighted avg	0.99	0.99	0.99	440

Figura 38. Desempeño general del modelo con datos nuevos.
Fuente: elaboración propia basada en la práctica.

Por otro lado, también se empleó la matriz de confusión (ver Figura 39) para identificar cuántas muestras de cada categoría fueron clasificadas incorrectamente por el modelo y a qué categorías fueron asignadas de forma errónea. Este análisis fue crucial ya que permitió analizar y determinar si los modelos desarrollados presentaban dificultades para clasificar alguna categoría en concreto. De este análisis, se pudo observar que la clase arroz tiene el menor porcentaje de precisión respecto a otras clases; no obstante, la misma supera el 80%, por lo que no supone un problema.

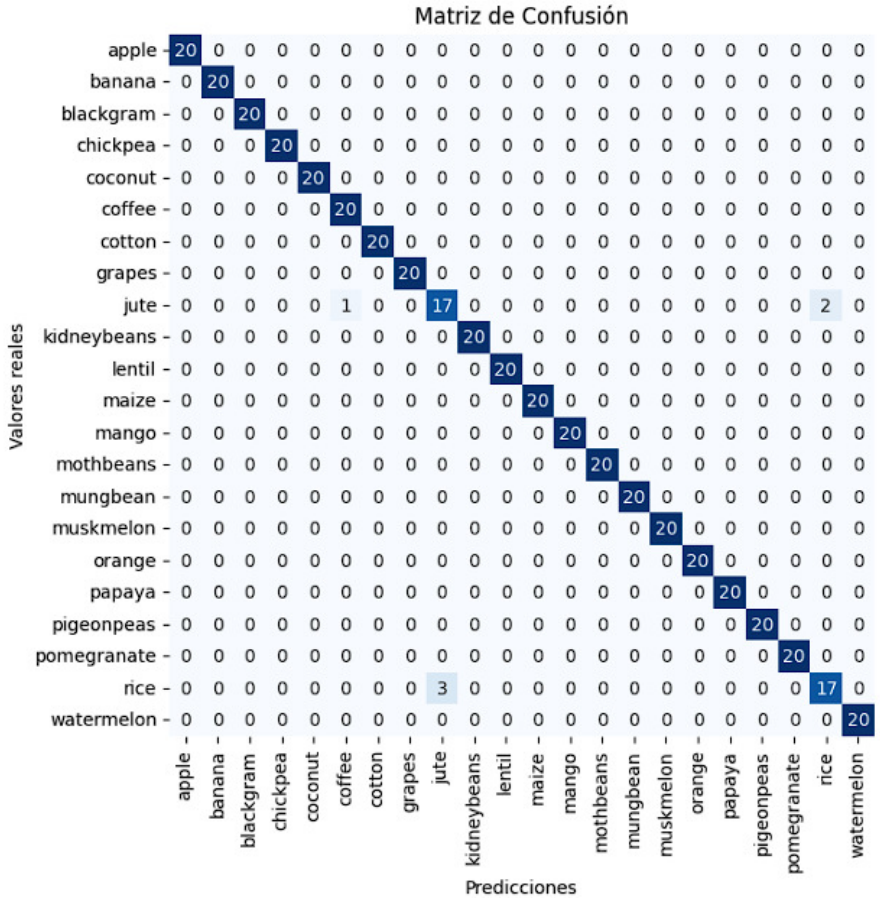


Figura 39. Matriz de confusión del modelo que emplea el algoritmo SVM.
Fuente: elaboración propia basada en la práctica.

5.1.3. Red Neuronal Densa

Para este caso, se validó el rendimiento general del modelo y el de cada categoría. Para el primer caso, se obtuvo como resultado una precisión (Accuracy) del 96% (ver figura 40). Mientras que para el segundo caso la precisión por categoría es la representada en la figura 41.

```
14/14 [==============================] - 0s 5ms/step
14/14 [==============================] - 0s 8ms/step - loss: 0.1213 - accuracy: 0.9568
Loss: 0.1213042289018631
Accuracy: 0.956818163394928
```

Figura 40. Accuracy de cada una de las categorías del modelo de datos.
Fuente: elaboración propia basada en la práctica.

```
rice: Precisión = 0.85
maize: Precisión = 0.95
chickpea: Precisión = 1.0
kidneybeans: Precisión = 1.0
pigeonpeas: Precisión = 1.0
mothbeans: Precisión = 0.85
mungbean: Precisión = 1.0
blackgram: Precisión = 0.9
lentil: Precisión = 1.0
pomegranate: Precisión = 1.0
banana: Precisión = 1.0
mango: Precisión = 1.0
watermelon: Precisión = 1.0
muskmelon: Precisión = 1.0
apple: Precisión = 1.0
orange: Precisión = 1.0
papaya: Precisión = 1.0
coconut: Precisión = 1.0
cotton: Precisión = 0.95
jute: Precisión = 0.8
coffee: Precisión = 1.0
grapes: Precisión = 1.0
```

Figura 41. Accuracy de cada una de las categorías del modelo de datos.
Fuente: elaboración propia basada en la práctica.

Nuevamente, se empleó la matriz de confusión (ver figura 42) para analizar el comportamiento del modelo y poder visualizar que categorías fueron clasificadas erróneamente. Como se realizó la codificación de números enteros en las categorías, no se visualiza la etiqueta como un string sino como enteros. A continuación, se detalla que números representan cada categoría: rice: 0: arroz, 1: maíz, 2: garbanzo, 3: frijol rojo, 4: guisantes, 5: frijol de rocío, 6: frijol mungo, 7: vigna mungo", 8: lenteja, 9: granada, 10: banana, 11: mango, 12: sandía, 13: melón, 14: manzana, 15: naranja, 16: papaya, 17: coco, 18: algodón, 19: yute, 20: café, 21: uvas.

A pesar que el modelo posee un menor porcentaje de precisión, que los modelos expuestos anteriormente, se puede concluir que tiene una buena capacidad para generalizar datos nuevos.

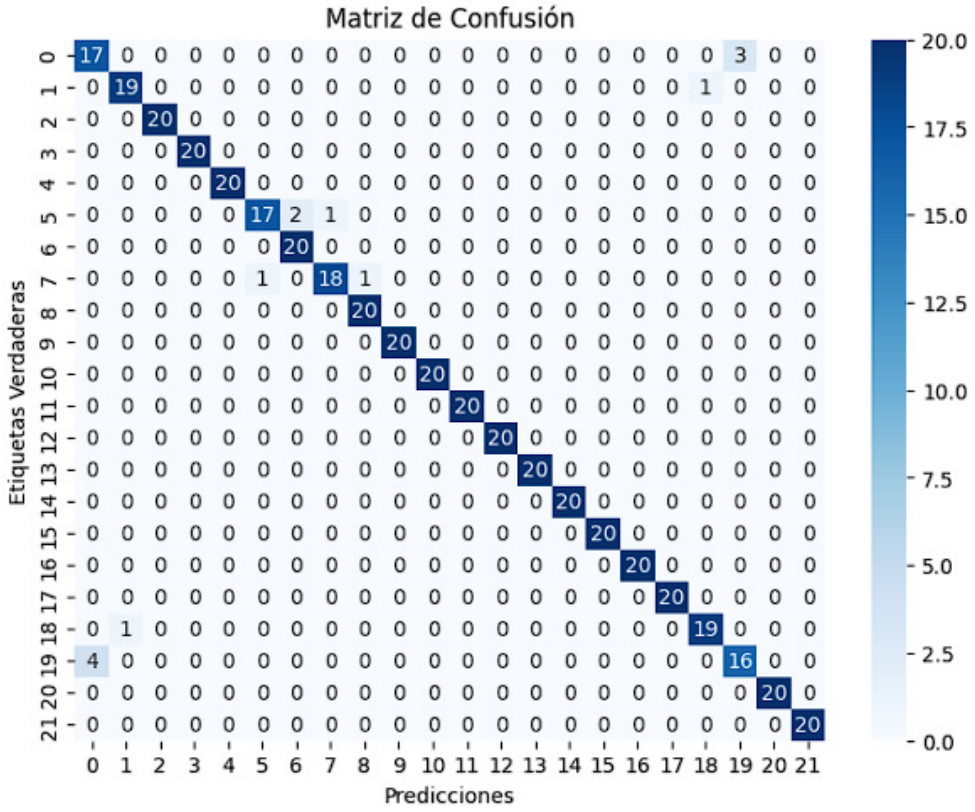
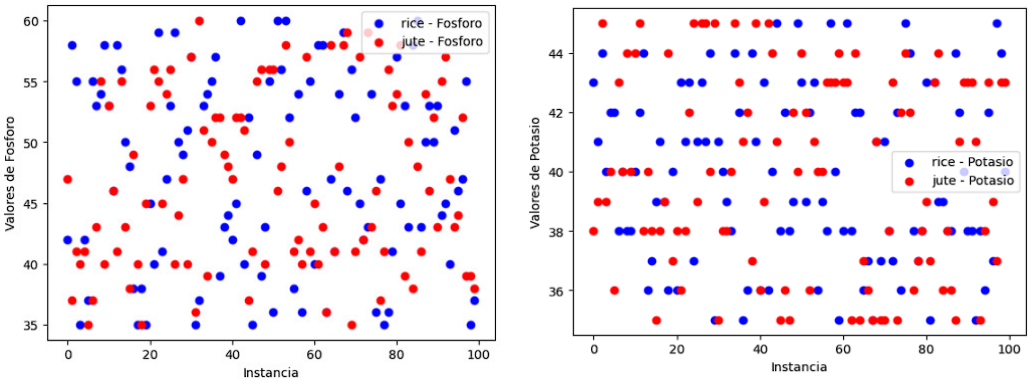


Figura 42. Matriz de confusión del modelo que emplea el algoritmo SVM.
Fuente: elaboración propia basada en la práctica.

Luego de analizar la matriz de confusión de los tres modelos se pudo observar que todos presentan cierta dificultad para clasificar las categorías arroz y jute. Es por ello, que se empleó gráficos de dispersión (ver figura 43) para representar la distribución de los diferentes atributos de las categorías mencionadas.



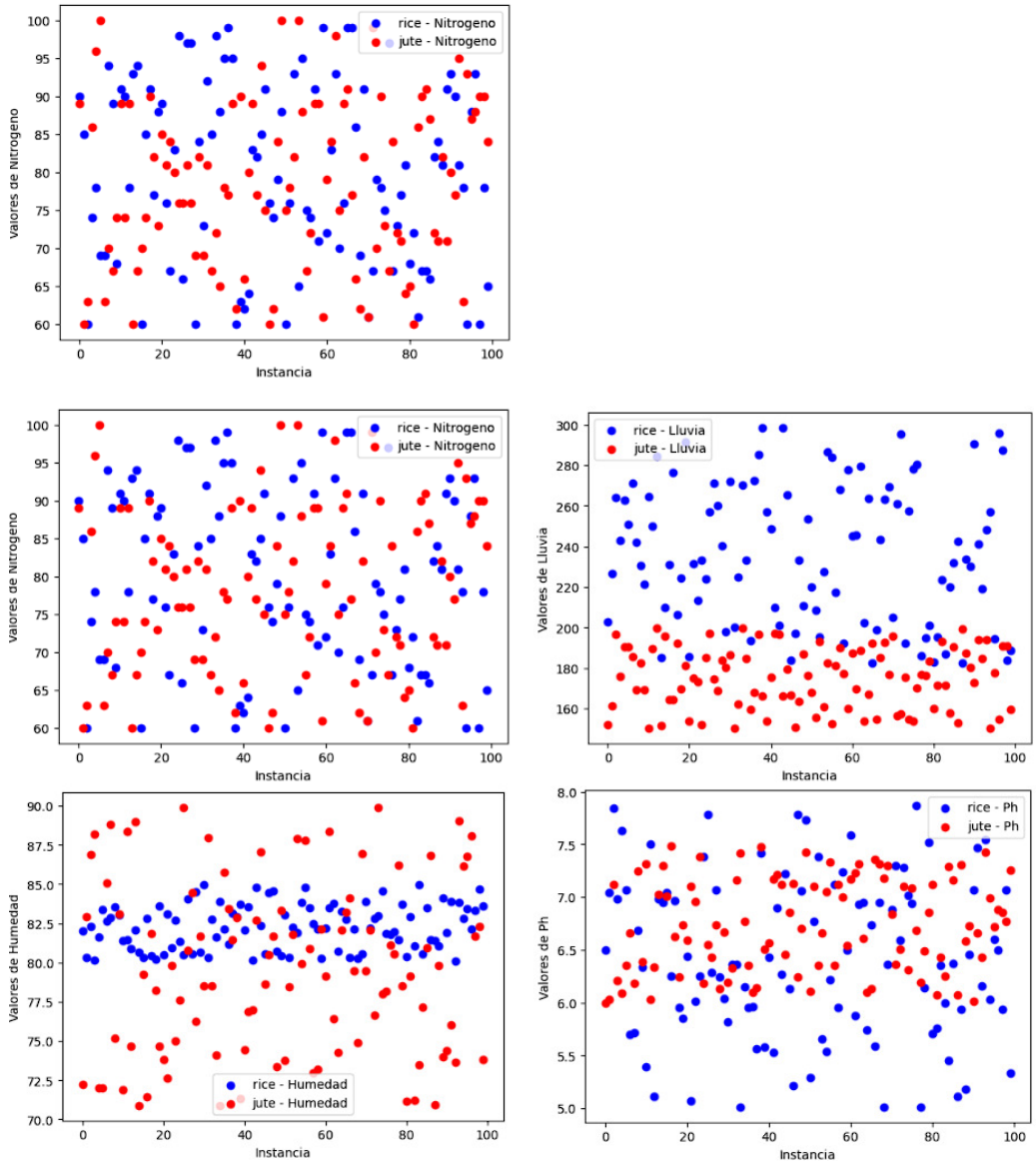


Figura 43. Gráficos de dispersión de los atributos nitrógeno, potasio, fósforo, temperatura humedad y lluvia de las categorías Rice y Jute.
Fuente Elaboración propia basada en la práctica.

En los gráficos de dispersión se pueden visualizar que las muestras de los atributos nitrógeno, potasio y fosforo comparten valores similares para ambas categorías, mientras que los casos de lluvia y humedad solo una fracción de los valores de las muestras se encontraron solapadas. Por otro lado, las muestras de temperatura se visualizan agrupadas entre los valores de 23 a 27, mientras que, para la categoría rice los valores están dispersos entre 20 y 27.

Teniendo en cuenta lo descripto anteriormente, los modelos presentan cierta dificultad para clasificar aquellas muestras cuyos atributos de humedad, pH, temperatura y lluvia se encuentran en el mismo rango, estos casos son difíciles ya que se encuentran dentro de la frontera de decisión entre una categoría y otra. Por otro lado, si se empleará el tamaño

completo del conjunto de datos, los modelos podrían tener una mayor cantidad de muestras de estos casos, lo que le permitiría contar con una mayor capacidad para poder clasificarlos.

5.2. Resultados del desarrollo de la aplicación

La aplicación cuenta con una Actitivity login, la cual se encarga de realizar la autenticación del usuario mediante el uso de Authentication de Firebase, y el proveedor que se empleó es Google. Cuando se inicia la aplicación, siempre y cuando no haya una sesión abierta, se abre un cuadro de diálogo que muestra todas las cuentas de Google, que se encuentran vinculadas al dispositivo (ver figura 44) y el usuario debe elegir con cuál desea loguearse. El objetivo de esta funcionalidad es poder determinar si el usuario posee una huerta asociada al email que empleó para autenticarse. Es por ello, que se consulta a Firestore si hay datos relacionados (huertas) a ese mail ya que, en caso de no haberlos, habrá funcionalidades que el usuario no podrá acceder, como ver los datos en tiempo real o ver qué cultivo es propicio para su huerta (ver figura 45).

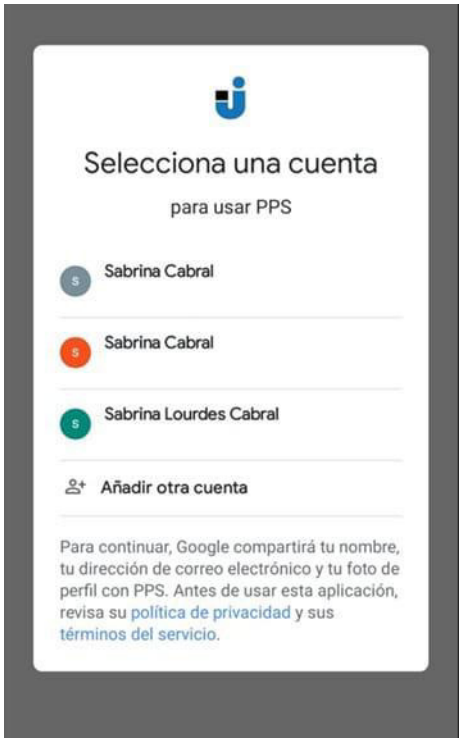


Figura 44. Inicio de sesión de la aplicación.
Fuente: Elaboración propia basada en la práctica.

Figura 45. Mensaje de error en la aplicación.
Fuente: Elaboración propia basada en la práctica.

Una vez concluido el proceso de inicio de sesión, se lleva a cabo la descarga del modelo para que pueda ser empleado posteriormente y, luego se inicia la actividad principal de la aplicación, llamada MainActivity y en ella se cargan los diferentes fragment que conforman a la aplicación. Por otro lado, en esta actividad también se gestiona el menú de navegación (figura 46), mediante el uso de NavGraph para definir y controlar de manera eficiente cómo se navega en toda la aplicación.

Figura 46. Menú desplegado de la aplicación.
Fuente: Elaboración propia basada en la práctica.

Por otro lado, la aplicación posee un menú con cuatro opciones, para que el usuario decida a qué pantalla navegar. Las mismas son las siguientes:

- Huerta: se carga como pantalla home luego del inicio de sesión y en ella se emplea el componente ViewPager, para que se puedan instanciar dos fragmentos. El primero permite a los usuarios ver en tiempo real el estado de las variables pH, temperatura y humedad de la huerta (ver Figura 47). En el segundo, se puede visualizar un histórico de los últimos 6 días de los datos de lluvia y humedad de la huerta. Para dicha visualización se utilizó un gráfico combinado de barras y líneas (ver Figura 48).

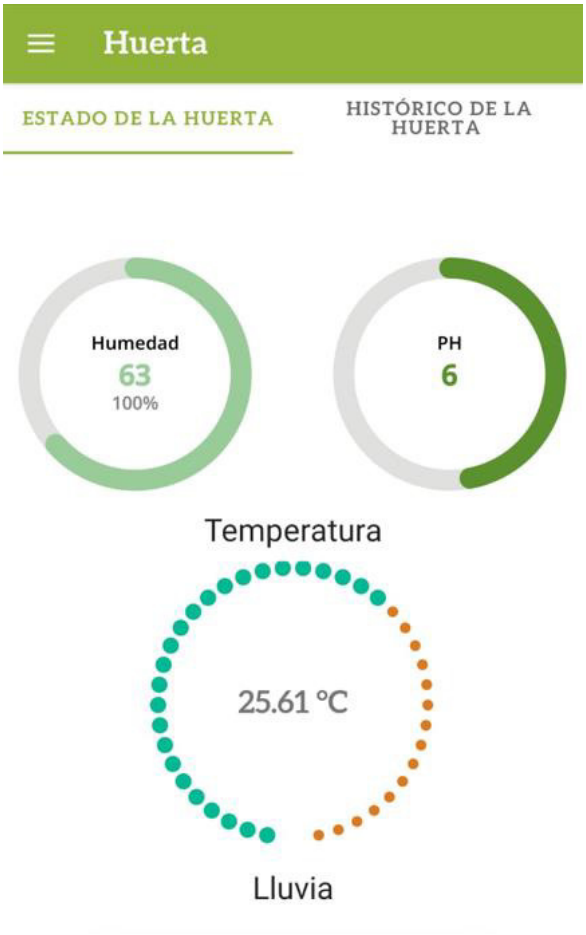


Figura 47. Estado en tiempo real de la huerta.
Fuente: Elaboración propia basada en la práctica.

Para crear estas representaciones visuales de la información, se emplearon las bibliotecas DonutGaugeView para los gráficos circulares y MPAndroidChart para el histórico.

Como los datos de temperatura, humedad, pH y lluvia son representados en diferentes pantallas se diseñó un layout genérico, que contenga todas las representaciones de dichos datos, con el objetivo de poder reutilizarlo en los layout de cada uno de los fragment que lo requieran. Por lo tanto, la clase Fragment solo se encarga de setear la información y en su layout asociado se incluye el genérico

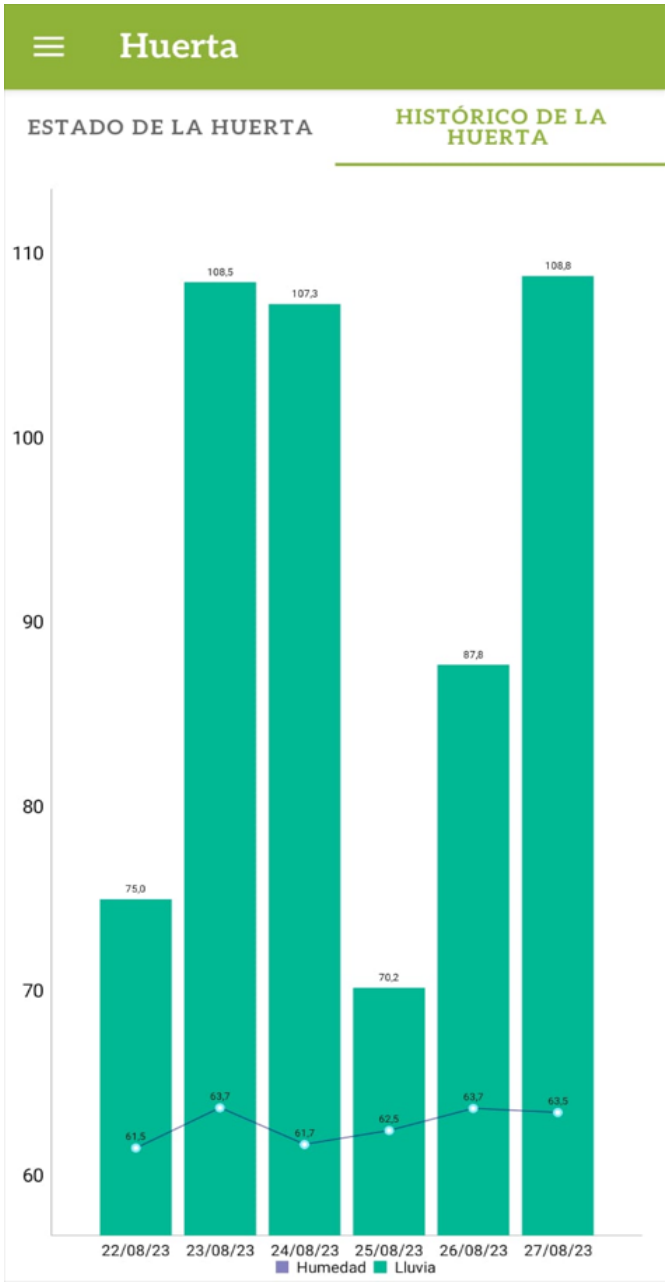


Figura 48. Representación gráfica del histórico de la huerta.
Fuente: Elaboración propia basada en la práctica.

- Tipos de cultivos: esta pantalla permite visualizar los diferentes cultivos que la aplicación puede clasificar. Posee un diseño maestro detalle, es decir, que se visualiza un recyclerView y luego de seleccionar algún ítem de él, se navega a otro fragment que puede ampliar la información del ítem que se seleccionó. Para poder cargar el componente recyclerView (ver figura 49), se empleó el ViewModel asociado al fragment, que se encarga de obtener los datos. Además, durante el tiempo que demora en obtenerlo, se visualiza una animación de carga y, una vez conseguido los datos, esta deja de ser visible para dar lugar al recyclerView.

Cada uno de los ítems que lo componen son seleccionables, por lo tanto, si se quiere obtener más información de algún cultivo en particular se debe hacer clic sobre ella, y mediante el navGraph se carga la pantalla del cultivo que se seleccionó (ver figura 50). En ella, se pueden visualizar los datos de la temperatura, humedad, pH y lluvia.

Figura 49. Pantalla de todos los tipos de cultivo que la aplicación puede clasificar.
Fuente: Elaboración propia basada en la práctica.

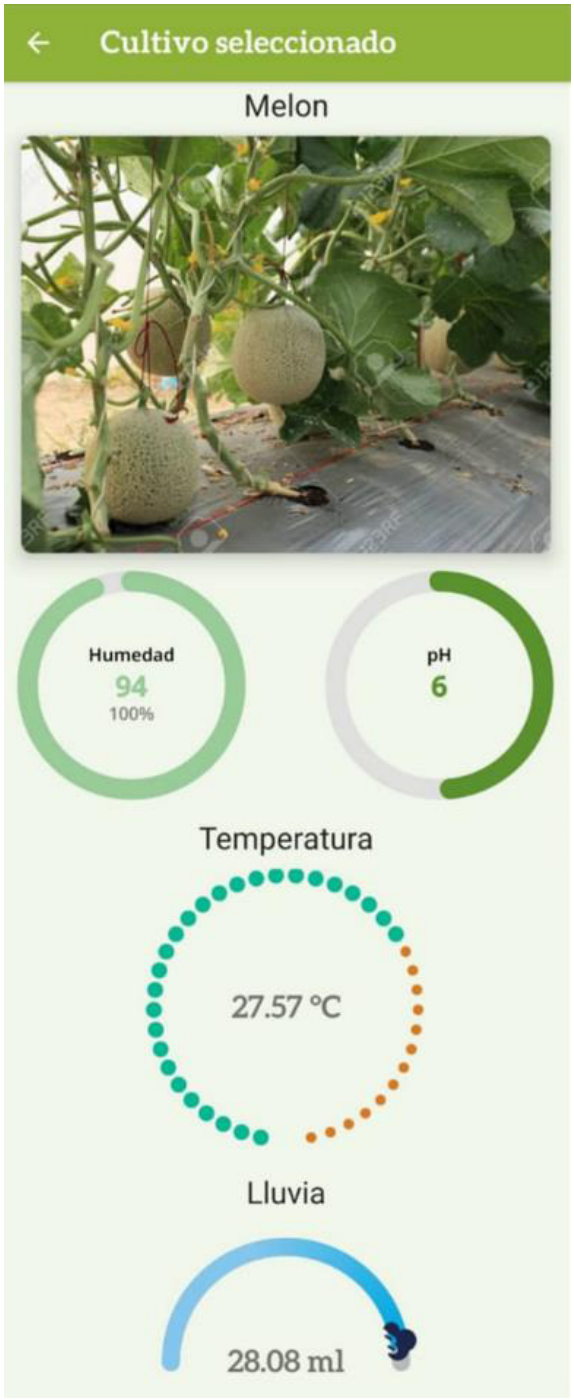


Figura 50. Detalle de un cultivo.
Fuente: Elaboración propia basada en la práctica.

- Cultivo clasificado: en esta sección, el ViewModel asociado a esta pantalla se encarga de realizar la llamada asincrónica para que, en base a los datos de la temperatura, pH, humedad, fósforo, nitrógeno y potasio de la huerta asociada al usuario que se logueó, el modelo determine qué cultivo es más propicio para su huerta. Una vez concluida la aserción, el ViewModel cambia el estado de los datos, y los observadores de las variables que se encuentran definidos en el fragment, reciben la notificación del

cambio de estado, por lo tanto, se produce una actualización en la UI (ver figura 51) con información concerniente al cultivo que el modelo determinó.

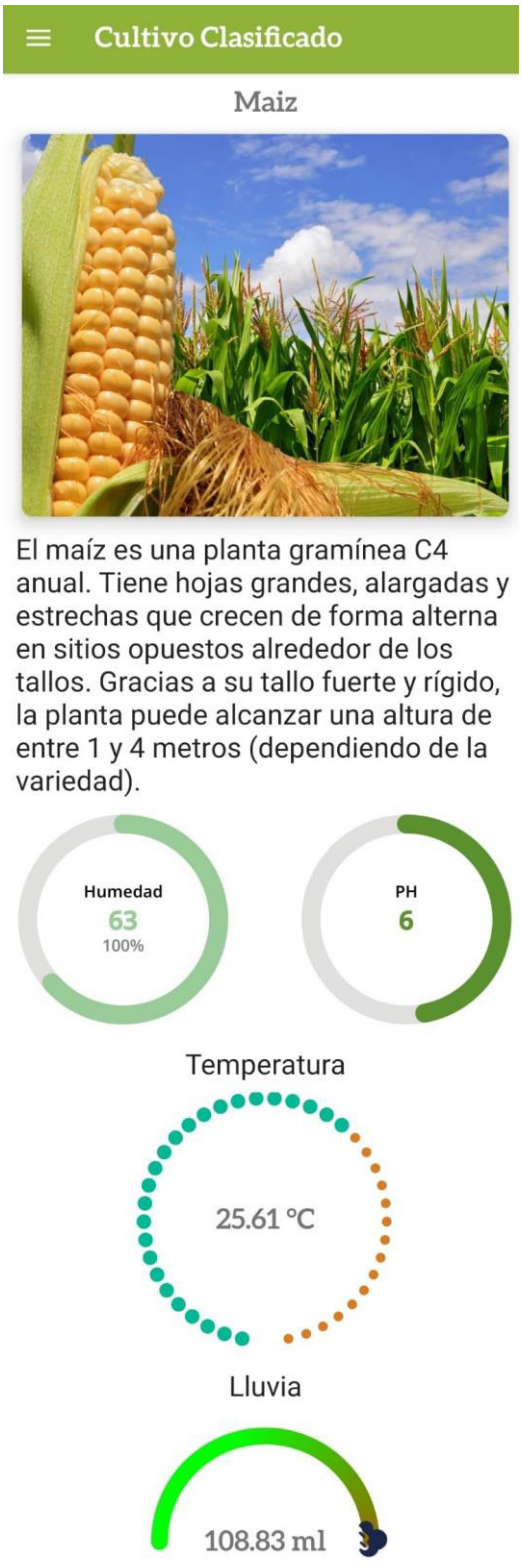


Figura 51. Información concerniente al cultivo clasificado.
Fuente: Elaboración propia basada en la práctica.

- Cerrar sesión: esta opción, como lo indica su nombre, tiene como objetivo cerrar la sesión del usuario y eliminar cualquier información concerniente a él dentro del almacenamiento local.

Conclusión

El desarrollo de esta Práctica Profesional Supervisada se estructuró en tres etapas que abordaron aspectos clave en el campo del aprendizaje automático y su integración con Android. La investigación inicial se centró en explorar diversos modelos para la clasificación de muestras, considerando opciones como regresión logística y SVM mediante la librería Scikit-learn.

La segunda etapa, de mayor duración, se enfocó en el desarrollo de los modelos y la optimización de sus hiperparámetros para garantizar un rendimiento óptimo, evitando el sobreajuste y manteniendo una buena capacidad de generalización. A pesar de que los dos primeros modelos alcanzaron un rendimiento satisfactorio, superando el 95% de precisión en la clasificación, surgió la necesidad de cambiar el modelo a uno creado en TensorFlow debido a la incompatibilidad entre la librería Scikit-learn con Android.

La transición a TensorFlow permitió la integración con la aplicación Android, aunque implicó enfrentar desafíos ya que, se utilizó la construcción de un modelo basado en redes neuronales. Este alcanzó una precisión del 95%, sin embargo, es ligeramente inferior a los modelos anteriores. El análisis de los mismos reveló una dificultad común en la clasificación de una categoría específica, debido a las similitudes en la distribución de algunos atributos entre las clases. Para poder suplir este inconveniente se podría emplear datos sintéticos para aumentar el conjunto de entrenamiento, mejorando así la capacidad de generalización del modelo.

Además, durante la evaluación de la aplicación, se identificaron oportunidades para mejorar su funcionalidad y utilidad. Un aspecto clave que podría fortalecerse es el desarrollo de una API, que facilite la obtención eficiente de los datos necesarios para el correcto funcionamiento de la aplicación. Este enfoque, optimizaría el rendimiento además de mejorar la seguridad y la escalabilidad de la misma.

Otro aspecto de mejora es la implementación de un motor de notificaciones. Esta característica sería invaluable para alertar a los usuarios sobre posibles anomalías en los parámetros de la huerta. La incorporación de notificaciones proporcionaría una capa adicional de información en tiempo real, permitiendo una respuesta proactiva ante situaciones críticas y mejorando la capacidad de monitoreo.

Adicionalmente, se podría implementar una funcionalidad que permita la segmentación de la huerta. Esta mejora proporcionaría una visión más detallada y específica del estado de diferentes áreas de la huerta, facilitando un monitoreo más preciso y

personalizado. La capacidad de segmentar la huerta sería especialmente valiosa para agricultores que buscan comprender y abordar problemas específicos en diferentes secciones del cultivo.

En conclusión, el trabajo abordó desafíos tanto en la selección y desarrollo de modelos de aprendizaje automático como en su integración efectiva con una aplicación Android. A pesar de las dificultades encontradas, se lograron resultados prometedores y se identificaron áreas de mejora, como la generación de datos sintéticos, que podrían impulsar aún más la capacidad predictiva del modelo. Este estudio contribuye al entendimiento práctico de la implementación de modelos de aprendizaje automático en entornos móviles y destaca la importancia de la adaptabilidad y compatibilidad entre librerías y plataformas.

Reflexión sobre la Práctica Profesional Supervisada como espacio de formación.

El desarrollo de la Práctica profesional supervisada supuso un desafío, debido a mi falta de conocimiento y experiencia en el ámbito de la inteligencia artificial. Esto implicó que se dedicaran extensas horas a la investigación del mismo y la creación de múltiples modelos para lograr cumplir con los objetivos propuestos. A pesar de que yo tengo un campo de dominio profesional definido, que es diferente al ámbito de la inteligencia artificial, la investigación mejoró mi capacidad de análisis, crítica y resolución de dificultades. Por otro lado, me permitió poner en práctica saberes previos adquiridos en mi trayectoria en la universidad al igual que en lo profesional. Finalmente, creo que también contribuyó a un mayor dominio en la realización de informes, ya que son pocas las materias en la que los realizamos.

Bibliografía

Alpaydin E., “Introduction to Machine Learning”. Second Edition (2010).

Android Developers. Arquitectura de capa de datos. Recuperado https://developer.android.com/topic/architecture/data-layer?hl=es-419. Fecha de consulta Julio de 2023.

Android Developers. Arquitectura de la capa de la UI. Recuperado https://developer.android.com/jetpack/guide/ui-layer?hl=es-419. Fecha de consulta Julio de 2023.

Android Developers. Capa de dominio. Recuperado https://developer.android.com/jetpack/guide/domain-layer?hl=es-419. Fecha de consulta Julio de 2023.

Android Developers. Descripción general de LiveData. Recuperado https://developer.android.com/topic/libraries/architecture/livedata?hl=es-419. Fecha de consulta Julio de 2023.

Android Developers. Principios comunes de arquitectura. Recuperado https://developer.android.com/jetpack/guide?hl=es-419. Fecha de consulta Julio de 2023.

Bishop C. M., “Pattern Recognition and Machine Learning”. First Edition (2006).

Burkov, A., “The Hundred-Page Machine Learning Book”. First Edition (2019).

Charu C., “Neural Networks and Deep Learning”. First Edition (2018).

DataScientest. Cross-Validation: definición e importancia en Machine Learning. Recuperado de https://datascientest.com/es/cross-validation-definicion-e-importancia. Fecha de consulta octubre de 2023.

El mundo de los datos. Codification one-hot. Recuperado https://elmundodelosdatos.com/tecnicas-para-codificar-variables-categoricas-ordinal-one-hot/. Fecha de consulta octubre de 2023.

Gerón A. “Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”. Second Edition (2019).

Harrigton, P., “Machine Learning in Action”. Manning, First Edition (2012).

Haykin, S., “Neural Network A Comprehensive Foundation”. Person, Second Edition (2001).

Hinge Loos in support vector machine. Hinge Loos in SVMs. Recuperado https://www.niser.ac.in/~smishra/teach/cs460/23cs460/lectures/lec11.pdf. Fecha de consulta octubre de 2023.

Japkowicz N. and Shah M., "Evaluating Learning Algorithms". First Edition (2011).

Molnar, C., Interpretable Machine Learning: A Guide For Making Black Box Models Explainable". Independently published (2022).

Müller A. C. and Guido S., "Introduction to Machine Learning with Python". First Edition (2016).

Pandas. Attributes and underlying data. Indexing, iteration. Recuperado https://pandas.pydata.org/docs/reference/frame.html. Fecha de consulta junio de 2023.

Rashka, S. and Mirjalilli, V., "Python Machine Learning". Packt Publishing, 2nd edition (2017).

Repetur A. (2019) "Redes Neuronales Artificiales". Universidad Nacional del Centro de la Provincia de Buenos Aires Facultad de Ciencias Exactas.

Russell, S. and Norvig, P., "Artificial Intelligence: A Modern Approach". Pearson, Third Edition (2009).

Scikit-learn. Computing cross-validated metrics. Recuperado https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score. Fecha de consulta octubre de 2023.

Scikit-learn. Cross-validation iterators with stratification based on class labels. Recuperado https://scikit-learn.org/stable/modules/cross_validation.html. Fecha de consulta octubre de 2023.

Scikit-learn. GridSearchCV, parameters, methods and Examples. Recuperado https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV. Fecha de consulta octubre de 2023.

Scikit-learn. Sklearn.model_selection.KFold. Recuperado https://qu4nt.github.io/sklearn-doces/modules/generated/sklearn.model_selection.KFold.html#sklearn.model_selection.KFold. Fecha de consulta octubre de 2023.

Scikit-learn. SVM, parameters, methods and Examples. Recuperado https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Fecha de consulta octubre de 2023.

Shawe-Taylor J. and Nello C., “Kernel Methods for Pattern Analysis”. Cambridge, First Edition (2004).

SparseCategoricalCrossentropy. Used in the notebooks. Recuperado https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy#used-in-the-notebooks. Fecha de consulta octubre de 2023.

Tensor Flow. Introduction to Tensors. Recuperado https://www.tensorflow.org/guide/tensor. Fecha de consulta junio de 2023.

Tensor Flow. Conversor de TensorFlow Lite. Recuperado https://www.tensorflow.org/lite/convert?hl=es-419. Fecha de consulta Julio de 2023.

Torres, J., “Deep Learning Introducción Práctica con Keras (primera parte)”. Marcombo, Primera Edición (2020).

Zheng A., “Evaluating Machine Learning Models”. First Edition (2019).