Tesinas de Grado

Matias Lorenzo Rodriguez

Machine Learning aplicado en huertas

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

Machine Learning aplicado en huertas

Rodriguez Matias Lorenzo

Florencio Varela, 12/2023

DATOS DEL ESTUDIANTE

Apellido y Nombres: Rodriguez Matias Lorenzo

DNI: 40947183

Nº de Legajo: 31155

Correo electrónico: matydarkar@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 43

PPS enmarcada en el artículo (4 ó 7) de la Resolución (CS) 103/16.

DOCENTE SUPERVISOR

Apellido y Nombres: Mg. Ing. OSIO, Jorge, Ing. Mauro Salina

Correo electrónico: josio@unaj.edu.ar, mdsalina@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Lic. y Prof. Bein, Paula Mariana

Correo electrónico: pbein@unaj.edu.ar

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Universidad Nacional Arturo Jauretche

Dirección: Av. Calchaquí 6200, Florencio Varela, (1888) Buenos Aires, Argentina

Teléfono: +54 11 4275 6100

Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Jorge Osio

Correo electrónico: josio@unaj.edu.ar

FIRMA DEL COORDINADOR DE LA CARRERA

Índice

1. Introducción

La PPS se desarrolló en el marco del proyecto de Investigación de la Universidad Nacional Arturo Jauretche UNAJ INVESTIGA 2020 (Código del Proyecto 80020200300027UJ y Resolución Rectoral N° 183/21 de fecha 29/06/2018), cuyo título es “Tecnologías de IoT y aprendizaje automático para la solución de problemas en el medio productivo y el cuidado del medio ambiente” dirigida por el Mg. Ing. Jorge Osio.

La propuesta de trabajo se enfocó en el desarrollo de una aplicación de software encargada de analizar una gran cantidad de datos almacenados en una base de datos provenientes de variables físicas medidas a los largo del tiempo en invernaderos. Luego se realizó el entrenamiento de una serie de algoritmos de aprendizaje automático para la posterior toma de decisiones en relación al estado del cultivo y el control de riego, iluminación y ventilación en función de los resultados.

El análisis de datos de forma automática brinda información relevante para la toma de decisiones y el control de determinados parámetros, de manera tal que un sistema sea capaz de conocer por sí solo el estado de su entorno e interactuar con él según la información recolectada.

Para llevar a cabo el desarrollo del algoritmo, se utilizó la Inteligencia Artificial (IA) para poder automatizar la toma de decisiones y la resolución de problemas. Esto implica utilizar técnicas de Machine Learning (Aprendizaje Automático) .

Los avances en IA han impulsado el uso de Big Data debido a su habilidad para procesar enormes cantidades de datos y proporcionar ventajas comunicacionales, comerciales y empresariales, que la han llevado a posicionarse como la tecnología esencial de las próximas décadas.

La IA está presente, por ejemplo, en los dispositivos cotidianos a través de bots o aplicaciones para móvil. El objetivo de cada una es hacer más fácil la vida de las personas.

1.3 Objetivos

1.3.1 Objetivos generales

El objetivo principal de la PPS fue desarrollar una aplicación a través de la cual se realice el procesamiento de datos provenientes de parámetros de un invernadero utilizando Machine Learning, aplicando al procesamiento de los datos, además de investigar el uso de la Inteligencia Artificial en ciencias de la computación.

1.3.2 Objetivos específicos

La idea principal es que, a partir de investigaciones y estudios especializados en la inteligencia artificial, se pueda llegar a desarrollar e implementar una solución para los cuidados de una huerta.

Para poder alcanzar ese objetivo principal y optimizar la gestión de un invernadero, es esencial abordar y prevenir diversas situaciones que pueden afectar el desarrollo de las plantas. Estas situaciones incluyen la escasez de nutrientes necesarios, la falta de riego o iluminación adecuados, así como las variaciones extremas de temperatura, entre otras variables críticas.

La implementación de un sistema de monitoreo ambiental se presenta como una solución integral. Este sistema se encargaría de medir y registrar las condiciones del hábitat, permitiendo anticipar y gestionar cualquier anomalía que pueda comprometer la integridad de las plantas. A continuación, se detallan los hitos necesarios para lograr nuestro objetivo principal:

- **Analizar los distintos tipos de aprendizaje automático e investigación de los diferentes usos de las mismas.** Fue indispensable comprender los conceptos básicos para poder desarrollar un sistema, por lo cual, la primera tarea consistió en la búsqueda de material bibliográfico para adquirir los conocimientos necesarios.

- **Analizar metodologías utilizadas para la toma de decisiones mediante procesamiento de parámetros en un invernadero aplicando Machine Learning.** El primer paso fue recolectar la información sobre los métodos utilizados para este tipo de tareas aplicados al contexto de un invernadero, teniendo en cuenta las distintas variables que tiene el mismo.
- **Crear un dataset (Conjunto de datos) de parámetros de huertas/invernaderos, que serán provistas a la red para su entrenamiento y posterior toma de decisiones sobre el accionamiento de los actuadores típicos para el riego e iluminación.** Para poder entrenar la red neuronal fue necesario la creación de un dataset adecuado con las variables que se pueden dar en un invernadero y las que eso conlleva. Para ello, fue necesaria una base de datos que contenga las imágenes que se utilizarán para el entrenamiento del sistema.
- **Desarrollar un modelo de aprendizaje automático capaz de analizar los datos de invernaderos y detectar anomalías y tomar decisiones en base de datos.** Luego se desarrolló un sistema capaz de aprender, procesar y tomar decisiones a partir de las imágenes que se le presentan. Lo cual ayudó a prevenir situaciones que pudieran poner en peligro a las plantas del invernadero.
- **Comparar modelos utilizados y determinar cuál es el más adecuado para este tipo de aplicaciones.** Así fue posible determinar cuál era el más eficiente para este tipo de aplicaciones teniendo en cuenta los resultados.
- **Testear la aplicación desarrollada y analizar los resultados.** Fue necesario comprobar que la aplicación sea capaz de obtener resultados correctos y que sea verdaderamente útil.
- **Probar la aplicación a través de un sistema embebido en un ambiente real.** Se corroboró que la aplicación no solo era capaz de funcionar en un sistema cerrado, con datos preparados, sino también en un ambiente real, con datos reales.

2. Marco teórico

2.1 Inteligencia Artificial

El concepto de IA surgió en la década de 1950 como consecuencia de intentar hacer que las computadoras fueran capaces de “pensar” por sí mismas. En un principio, consistía en sets de instrucciones gigantes que permitían a las computadoras resolver problemas complejos pero no podían “aprender” a solucionar nuevos problemas a partir de los anteriores.

Esta incapacidad de aprender por sí mismas que tenían las inteligencias llevó al surgimiento de Machine Learning (ML) y Deep Learning (DL).

Ambos son subcampos de la inteligencia artificial (aunque se suelen confundir y pensar que son lo mismo). “Deep Learning” es un subcampo de ML y, a su vez, ML es un subcampo de la inteligencia artificial, como se muestra en la figura 1:

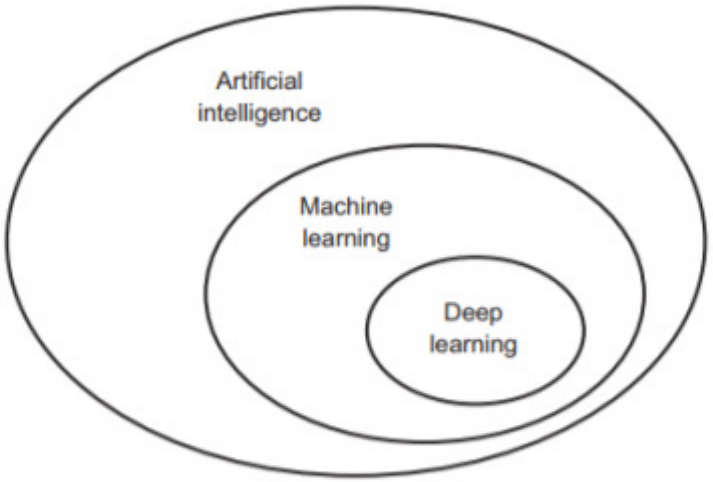

Figura 1. Inteligencia artificial, Machine Learning y Deep Learning
Fuente: Recuperado de https://i2.wp.com/cdn-images-1.medium.com/freeze/max/1000/1*IoCK5tsGd59R66TO0zOQRA.png?w=1170&ssl=1

2.2 Machine Learning

Machine Learning (ML) es un paradigma de programación que surge como respuesta a si una computadora es capaz de aprender por su cuenta cómo resolver nuevos y complejos problemas sin la necesidad de que un programador escriba las instrucciones que debe seguir para llegar a una respuesta correcta. Esta rama de la programación está directamente relacionada con la estadística, ya que lo que se logra es que una computadora sea capaz de detectar patrones estudiando los datos proporcionados por el programador. De esta forma se puede predecir resultados e identificar resultados inesperados o errados. Pero ¿cómo logra una computadora aprender a reconocer patrones?

En la programación clásica, un desarrollador escribe las *reglas* que debe seguir la computadora para obtener unos resultados. Pero en ML, funciona al revés ya que se busca encontrar esas *reglas* a partir de un set de respuestas. Se presentan una gran cantidad de datos para instruir al sistema a reconocer patrones entre aquellos y así llegar a las reglas pertinentes. Esto es posible gracias a la capacidad de cálculo que tienen las computadoras que procesan las grandes cantidades de datos que se requiere para que el sistema de ML aprenda a identificar los patrones entre todos los datos. Para lograr esto hay varios tipos de ML, las más conocidas son: el aprendizaje supervisado y el no supervisado.

2.2.1 Aprendizaje supervisado

Éste se basa en entrenar al algoritmo con un set de datos que incluyen las respuestas, comúnmente llamadas *etiquetas* (*labels* en inglés).

Un ejemplo de este tipo de aprendizaje sería un programa para pasar de grados Celsius a Fahrenheit. En la programación tradicional, lo normal sería una función que reciba por parámetro los grados C°, lo multiplique por 1.8 y le sume 32 para obtener el equivalente en F°.

Esto funciona diferente en Machine Learning. Lo principal es tener dos conjuntos de datos, uno con los valores principales y otros con las respuestas, uno con los grados celsius y otro en grados fahrenheit. Ver la siguiente tabla:

Grados Celsius	Grados Fahrenheit
0	32
32	89,6
100	212
58	136,4
10	50
86	186,6

A partir de esta tabla de conversiones, se intenta “entrenar” al algoritmo, ajustando el coeficiente del mismo para que pueda estimar una respuesta lo más cercana posible a los valores originales. Al final, el algoritmo será capaz de calcular otros datos que no aparezcan en el set de datos original.

Hay varios algoritmos que pertenecen a este tipo de ML, como pueden ser: *k-Nearest Neighbor*, *Linear Regression*, *Logistic Regression*, entre otros.

2.2.2 Aprendizaje no supervisado

Acá los datos no están etiquetados sino que se utiliza un algoritmo para separarlos y agruparlos según tengan cosas en común. Por ejemplo, en una página web de música se detecta que los hombres de entre 16 - 18 años prefieren escuchar rock nacional mientras que los de entre 40 - 50 años escuchan rock de los 80 's. Estás agrupaciones se conocen como “clusters” y el algoritmo las forma sin ayuda, sólo requiere encontrar las conexiones entre los distintos tipos de datos.

2.2.3 Aprendizaje por esfuerzo

En está metodología, el algoritmo intenta encontrar los resultados correctos sin saber cuál es la respuestas. Para esto, un “agente” interactúa con el entorno para poder alcanzar el resultado correcto. De ser así, obtiene una recompensa o positiva o una negativa. Por cada ciclo en este bucle, el agente va a cambiar de estado, en el que actualizará su conocimiento y así sucesivamente

2.3 Deep Learning

Deep learning (DL), o “aprendizaje profundo” en español, es un subcampo de Machine Learning, comúnmente utilizado para el reconocimiento de imágenes, voces o inclusive para predicciones. El término profundo viene por las capas de procesamiento que lo componen y que son las responsables de que la máquina aprenda a través de los datos que le son administrados.

DL implementa un sistema de capas compuestas por “neuronas”, llamado “redes neuronales”, cuyo nombre viene de la biología debido a que el funcionamiento es similar. Se busca que el algoritmo sea capaz de aprender de forma automática; gracias a las capas que componen la red que, al ser alimentadas por enormes cantidades de datos, pueden aprender de ellos.

La capa de entrada se encarga de recibir los primeros datos, procesarlos y enviarlos a los nodos que componen las capas ocultas, o “hidden layers”, en inglés. Dichos nodos procesan nuevamente la información y la van enviando a los subsecuentes nodos pertenecientes a la misma sección, según corresponda, hasta llegar a la última instancia (“la capa de salida”). Una vez ahí, los datos son tratados una vez más y ,así, generan un resultado. Todo esto es lo que compone una red neuronal; como puede verse cómo se muestra en la figura 2:

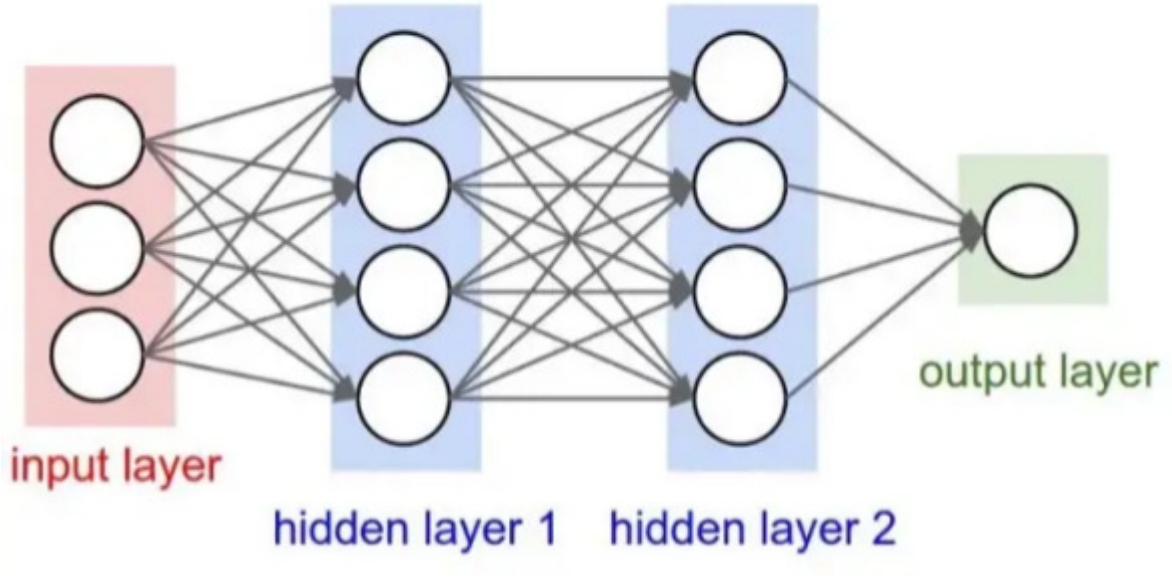

Figura 2. Modelo de algoritmo de Deep Learning.
Fuente: Recuperado de
https://towardsdatascience.com/building-a-deep-learning-model-using-keras-1548ca149d37

Al principio, las capas que componen la red tienen un parámetro (conocido como “peso”) con un valor inicial asignado al azar. Para poder aprender, ellas reciben grandes cantidades de datos que son procesados para poder ajustar los pesos de cada neurona y envíar el resultado a la siguiente capa, y así sucesivamente, hasta obtener un resultado final.

Es posible que una red neuronal esté compuesta por cientos de capas, por lo que se puede llegar a tener cientos de miles de parámetros. Entonces, encontrar un valor correcto para todas puede parecer imposible, sobre todo pensando que el cambio de una capa puede afectar a la siguiente y, por consiguiente, a toda la red.

Cuantas más capas compongan la red neuronal, más acertado será el resultado final; razón por lo cual es necesario también tener una gran cantidad de datos para entrenar cada una. Por esto, es necesario contar con una gran capacidad de procesamiento de GPU para ser capaz de procesar todos los datos que requiere una red y así llegar a los valores correctos, o los más aproximados, en todas y cada una las capas.

2.3.1 Diferencias con ML

Tanto ML como DL intentan imitar la forma en la que aprende el cerebro humano a distinguir patrones, sólo que a un nivel mucho mayor al ser capaces de analizar miles de datos en cuestión de segundos. A pesar de esto, no son iguales debido a que difieren en su forma de aprendizaje automática.

En ML, para que aprenda, es necesario guiar a la máquina en cada paso del proceso. Si comete un error, hay que señalarlo. En DL, la máquina hace todo lo anterior por sí misma, ante cada nuevo dato que recibe.

En ML, las principales características de los datos de entrada son seleccionadas manualmente y clasificadas por un tercero. En DL, en cambio, los pasos de extracción de características y clasificación son totalmente automáticos.

Por ejemplo, se utilizan ambas metodologías para distinguir si un animal es un perro o no. En ML, se extraen rasgos característicos del animal (la forma del cuerpo, cabeza, piernas, ojos, entre otras cosas). Luego, el algoritmo los clasifica como un perro o no. En DL, todo este proceso lo hace el algoritmo por su cuenta, cómo se puede ver en la figura 3:

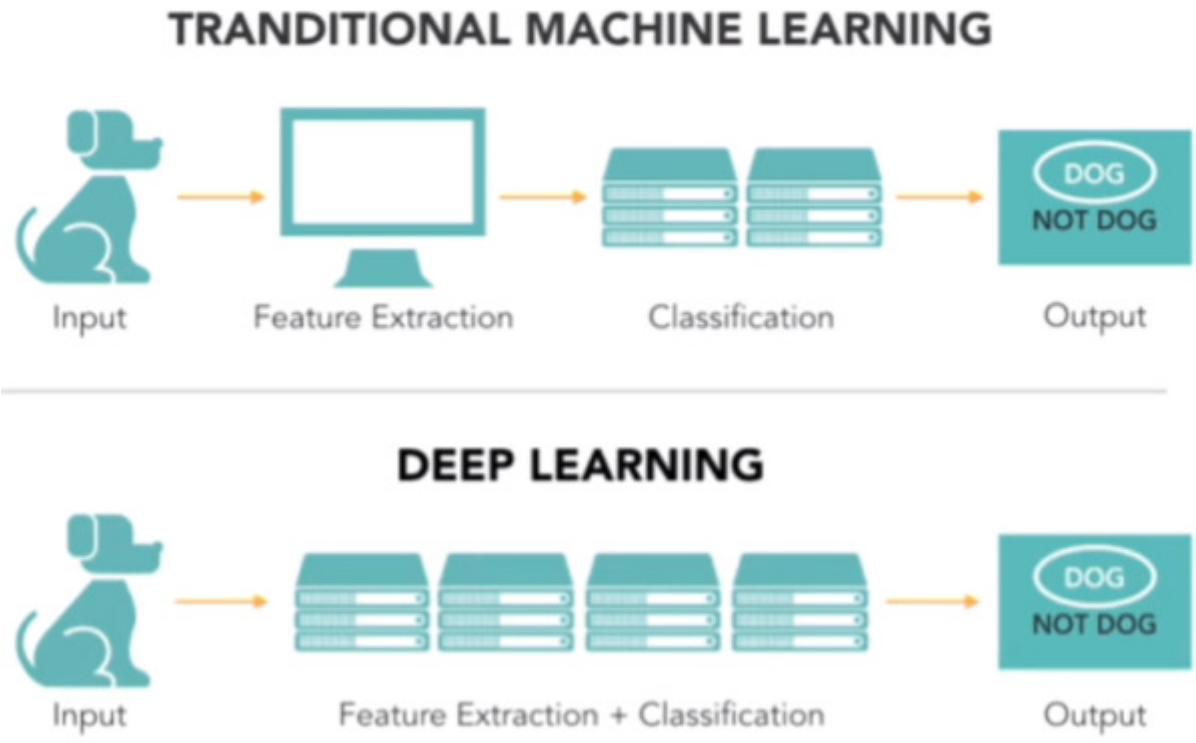

Figura 3. Diferencia en el funcionamiento entre machine learning y deep learning. Fuente: Recuperado de https://bigdatapath.wordpress.com/2019/09/13/ai-vs-machine-learning-vs-deep-learning-whats-the-difference/

2.4 Redes neuronales artificiales

Las redes neuronales artificiales son un método de IA para enseñar a las computadoras a imitar la forma en la aprende el cerebro humano. Están compuestas por tres tipos de capas: la de entrada (*input layer*), una o varias ocultas (*hidden layers*) y una de salida (*output layer*).

La primera es la que recibe la información del exterior, analiza los datos, los procesa o clasifica y, posteriormente, los pasa a la siguiente. Las segundas toman los datos de entrada o los que reciben de las capas ocultas anteriores y cada una se encarga de analizar y procesar aún más dichos datos. Por último, la tercera muestra el resultado final del procesado de los datos realizado por la red. Esta puede tener uno o varios nodos dependiendo de cómo se clasifique el resultado, por ejemplo, si el resultado final es binario (es decir 1 o cero) con uno solo es suficiente.

Todas están compuestas por neuronas (o nodos) las cuales tienen asociado un peso y una función de activación. Dichos nodos reciben un estímulo, lo procesan y envían una respuesta al siguiente.

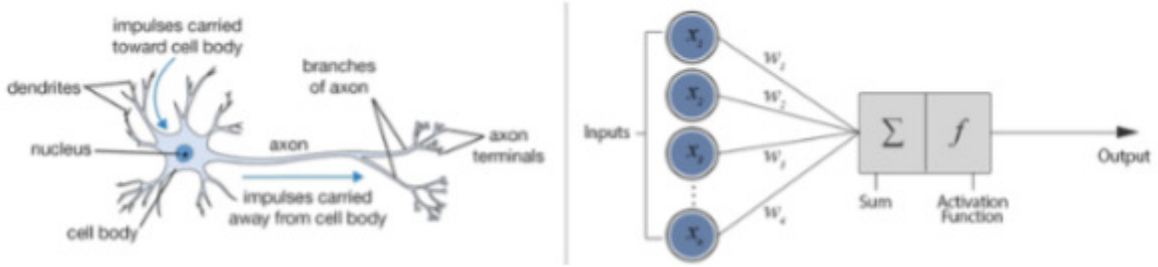

Figura 4. Comparación de una neurona biológica con una artificial.
Fuente: Recuperado de https://www.datacamp.com/tutorial/deep-learning-python

La neurona artificial recibe un valor X_i a la que se le asocia un peso W , donde la sumatoria de datos X multiplicados por su respectivo peso da un valor que será utilizado en la función de activación, cuyo resultado será el valor enviado a la siguiente neurona. Siendo la ecuación final que se lleva a cabo en cada neurona la siguiente:

$$f(X) = b + \sum_{i=1}^{n} wi \cdot xi$$

(ec. 1)

2.4.1 Sesgo

El sesgo (*bias* en inglés) actúa como una “falsa neurona” y representa la variable *b* en la ecuación anterior (ec. 1) y controla qué tan dispuesta está una neurona a generar una entrada, siendo los valores predeterminados 1 o un 0, sin importar los pesos. Un sesgo alto hace que la neurona requiera una entrada más alta para generar una salida de 1; uno bajo lo hace más difícil. Esto sirve para mover la curva de la función a la izquierda o a la derecha, según convenga para un mejor aprendizaje.

2.4.2 Función de activación

La función de activación o umbral se encuentra a la salida de la neurona y procesa -una vez más- los datos antes de enviarlos a la siguiente capa. Utiliza el resultado de la sumatoria anterior y le realiza una transformación, cuyo resultado determinará cuáles serán las neuronas de la siguiente capa en activarse.

Una red neuronal puede modelar cualquier tipo de función desde funciones lineales a funciones no lineales. Y, si bien hay otras, algunas de las más usadas son las siguientes:

2.4.2.1 Función lineal

Las funciones lineales, o de identidad, permiten que el resultado de la salida pueda ser igual que lo de la entrada; por lo que al aplicarse en una red con varias capas se dice que es una *regresión lineal*.

Fundamentalmente, se utiliza para predecir el valor de un resultado basado en el valor de otra variable, es decir, se puede estimar un valor X a partir de una variable independiente Y. Un uso común que tiene este tipo de funciones puede ser predecir algún valor numérico.

Su función es la siguiente y el resultado gráfico que genera se puede ver en la figura 5:

$$f(x) = x$$

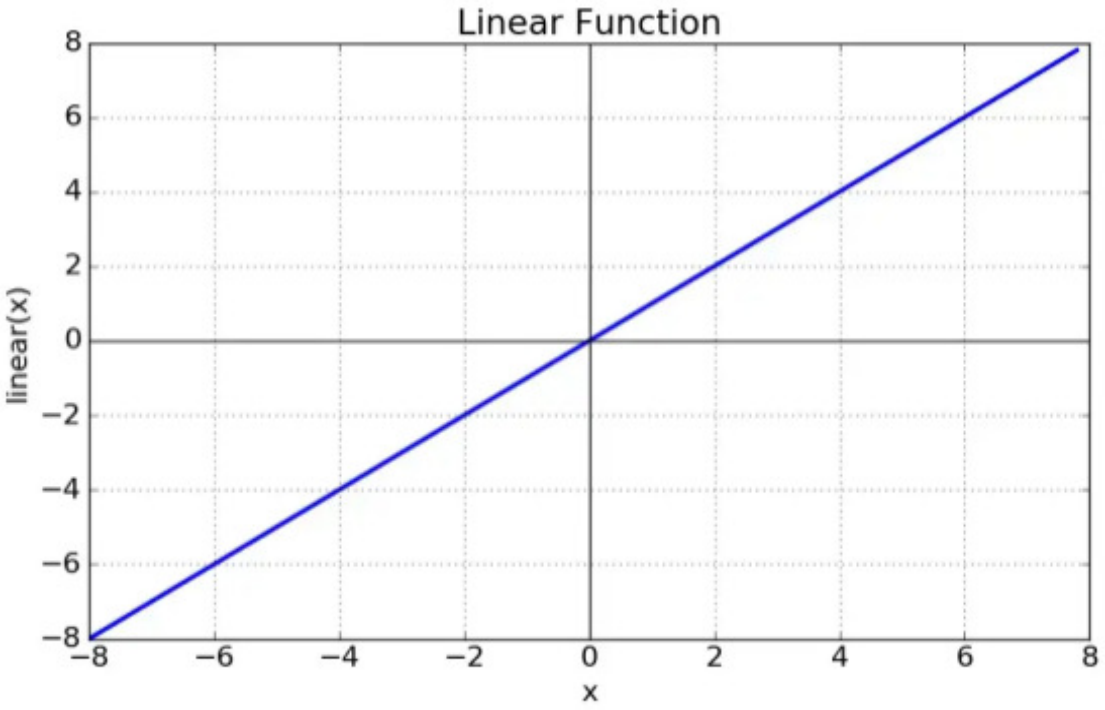

Figura 5. Función lineal.
Fuente:
https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.4.2.2 Funciones no lineales

Estas son las que usan los algoritmos de deep learning al poder ajustarse mejor a problemas más complejos. Hay varios tipos:

2.4.2.2.1 Función umbral

La umbral (o escalón) indica que, si el resultado es menor que 0, la salida será 0. De ser mayor o igual a 0, la salida será 1. Son utilizadas para clasificar o cuando se tienen resultados categóricos.

Su función es la siguiente y su resultado gráfico se puede ver en la figura 6:

$$f(x) = 0\, para\, x < 0 \cap 1\, para\, x \geq 0$$

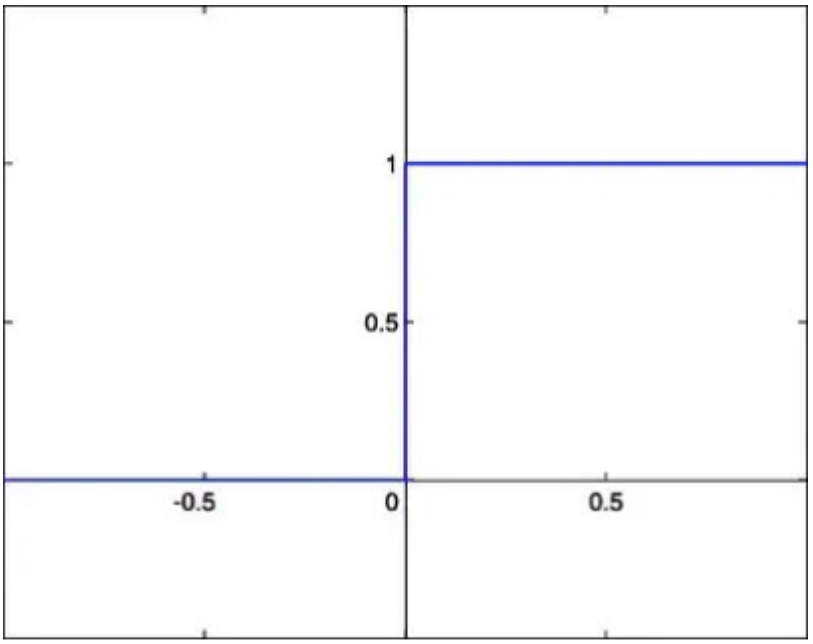

Figura 6. Función umbral.
Fuente:
https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.4.2.2.2 Función sigmoide

Las sigmoides (o logísticas) tienen como resultado un número entre el 1 y el 0, es decir, este rango proporciona una determinada probabilidad.

En este modelo probabilístico, al utilizar valores negativos muy bajos, el resultado será menor que 0. Si la entrada es 0, la probabilidad será de un 50%. Entonces, cuanto más elevada sea la entrada, más aproximado al 100% de probabilidad.

Tiene una forma como la que se ve en la figura 7 y la función es:

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

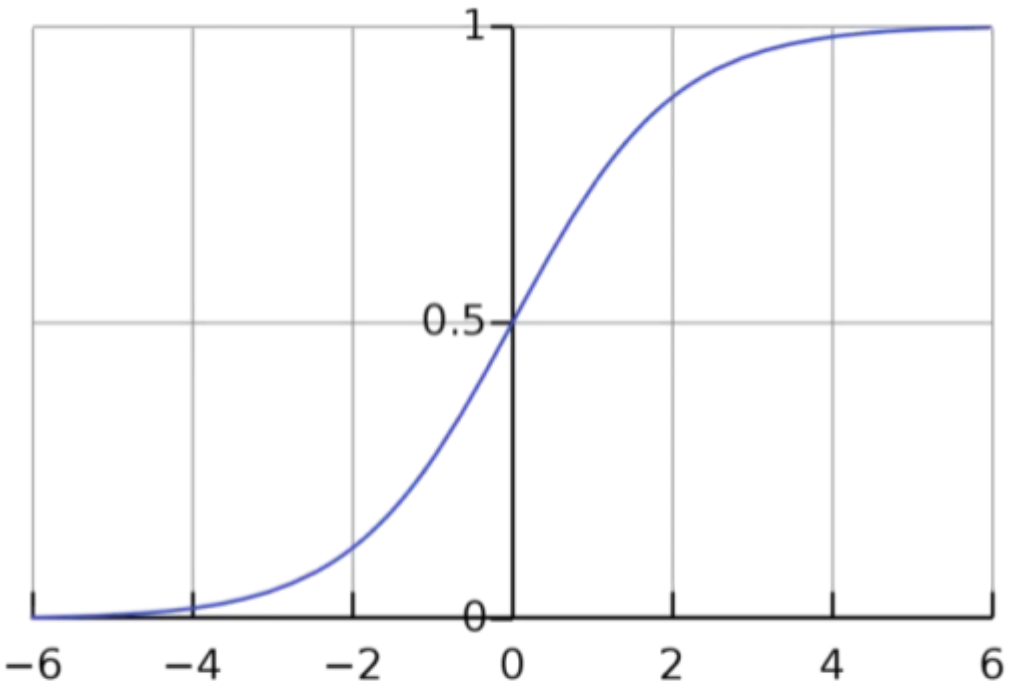

Figura 7. Función sigmoide.
Fuente:
https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.4.2.2.3 Función tangente hiperbólica

La tangente hiperbólica, conocida como de activación, puede tomar valores de entre -1 y 1. Sufre del mismo problema que la sigmoide por la falta de gradiente, a pesar de estar centrada. Dicho inconveniente puede producir errores con el algoritmo de propagación hacia atrás, afectando a las demás capas.

El gráfico de este tipo de funciones es como el que se puede ver en la figura 8 o similares, y la función tiene está fórmula:

$$f(x) = \frac{1}{1 + e^{-2x}} - 1$$

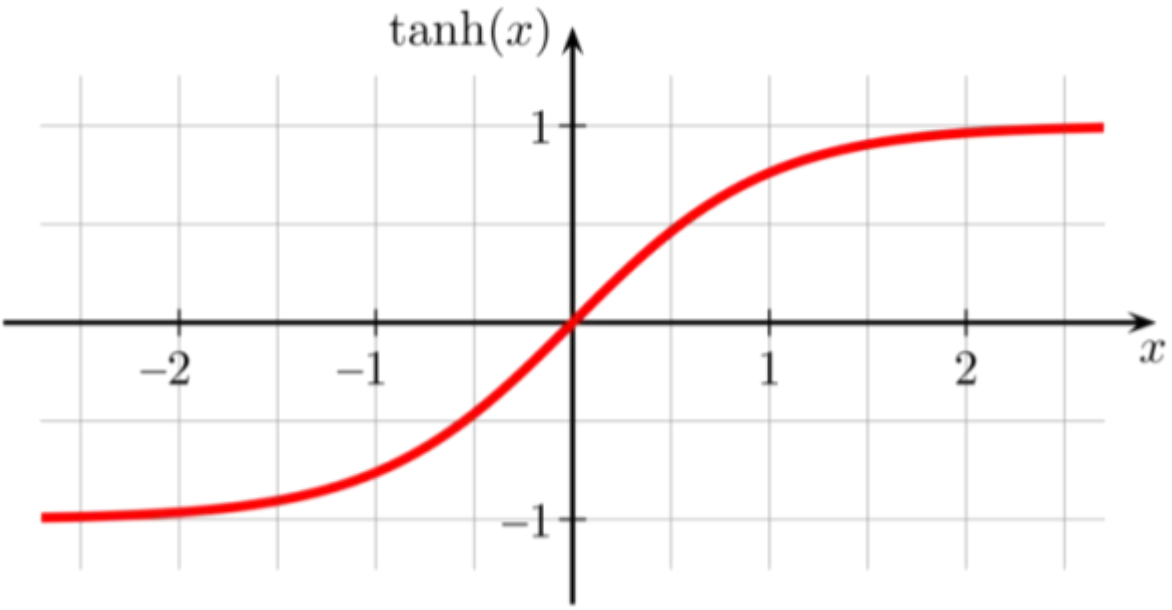

Figura 8. Función tangente hiperbólica.
Fuente:
https://bootcampai.medium.com/redes-neuronales
-13349dd1a5bb

2.4.2.2.4 Función ReLu

Las ReLu (Rectified Linear Unit o Unidad linear rectificada) son las más utilizadas debido a que permiten el aprendizaje gracias a que la derivada de las mismas permiten la propagación hacía atrás de los resultados. Además, no es necesario que se activen todas las neuronas, por lo tanto, es más efectiva que otro tipo de funciones.

Si recibe valores de entrada negativos muy altos, el resultado será cero. En cambio, si son positivos, quedará igual. Además, el gradiente será 0 en el segundo cuadrante y 1 en el primero.

Si el resultado es menor que 0, la neurona se desactiva y no puede cambiar ese estado, aunque hay una versión de este algoritmo que previene esto llamada *Leaky ReLU*.

La función tiene una forma como la que se ve en la figura 9 y su función es la siguiente:

$$f(x) = máx(0, x)$$

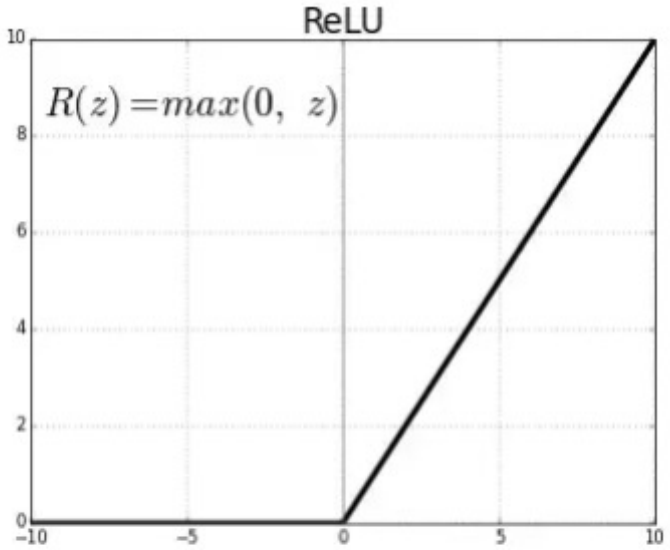

Figura 9. Función ReLU.
Fuente:
https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.4.2.2.5 Función Softmax

Este tipo se parece a las logísticas pero es utilizada normalmente para la clasificación de datos. Por ejemplo, si recibe como entrada la imagen de una fruta cualquiera y se solicita saber qué tipo de fruta es, la red proporcionará distintas posibilidades siendo estás 0,1 o 10% de probabilidades de ser una manzana, 0,25 o 25% de ser una naranja, etc. La suma de todas las probabilidad siempre es 1.

La forma de la función y su respectiva fórmula se pueden ver en la figura 10.

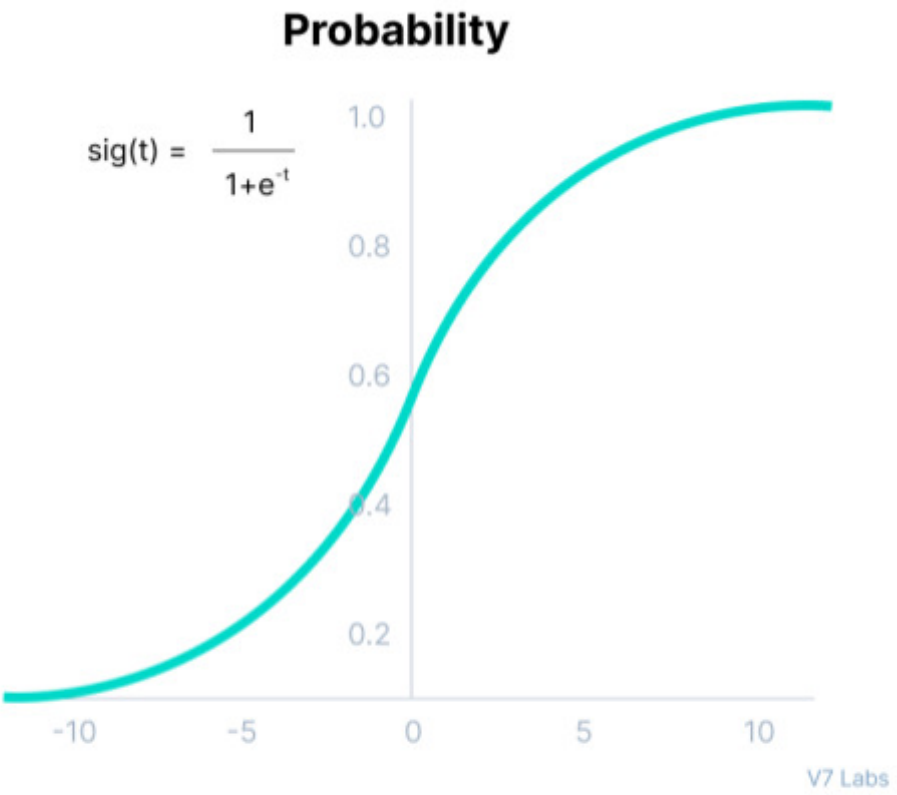

Figura 10. Función Softmax y su respectiva fórmula.
Fuente: https://www.v7labs.com/blog/neural-networks-activation-functions#:~:text=The%20linear%20activation%20function%2C%20also,the%20value%20it%20was%20given.

2.4.3 Proceso de aprendizaje

Tanto en el proceso en las redes neuronales artificiales como en las biológicas, a mayor repetición del proceso mayor el aprendizaje. Como consecuencia de lo anterior, mejores serán los resultados que se obtendrán.

Por ejemplo, se busca una red neuronal con dos nodos que sea capaz de aprender a imitar un sistema de calificaciones, dicho de otro modo, que pueda ponderar una nota final a partir de dos exámenes. Al principio, como la red no conoce los resultados, se inicia con un peso de 0,5 en los nodos, y empezando el entrenamiento ingresan las notas 5 y 9. El resultado ideal sería un 7 pero la red devuelve un 6,2. Esto no quiere decir que no funciona, sino que necesita más repeticiones. A la siguiente iteración, los pesos cambian: 0,4 y 0,6. En está iteración el resultado se acerca más,

ahora será un 6,8. Así hasta tener el resultado más cercano al valor correcto.

Este proceso de aprendizaje es un proceso que se repite hacia adelante (*Forward Propagation*) y hacia atrás (*Backward Propagation*) a lo largo de toda la red neuronal. Estás dos partes, en conjunto con la función de pérdida, son las partes que componen al aprendizaje.

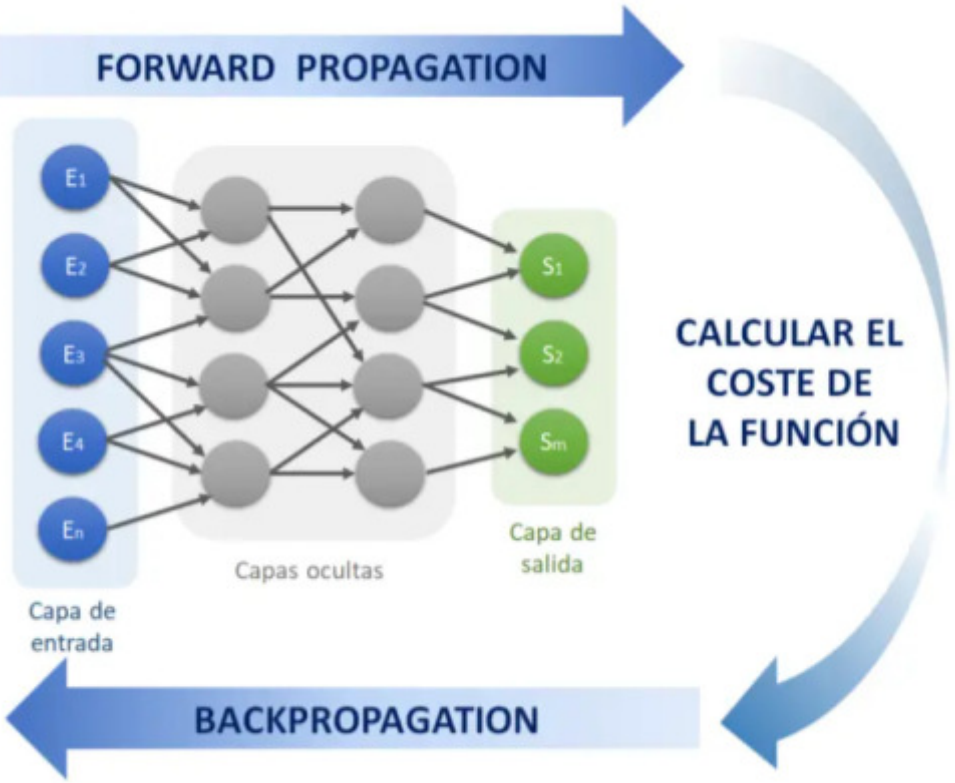

Figura 11. Proceso de aprendizaje de una red neuronal.
Recuperado de: https://www.diegocalvo.es/definicion-de-red-neuronal/

2.4.3.1 Forward propagation

Esta primera parte del proceso se inicia en la capa de entrada, pasa por todas las ocultas y termina en la de salida.

Aquí, cada neurona realiza una suma ponderada de todas las entradas de acuerdo al peso de cada una. El resultado de dicha sumatoria se traslada a la función de activación.

Por último, envía el resultado a la siguiente capa y así sucesivamente por toda la red.

2.4.3.2 Función de costo

Una vez finalizada la primera etapa, se realiza la función de costo, o pérdida. Indica el grado de precisión que tiene la red para realizar predicciones para una determinada entrada.

En otras palabras, está función indica si la respuesta final de la red está muy alejada de la deseada. La fórmula de dicha función (figura 12) es una sumatoria del cuadrado de todas las diferencias dividido el número de neuronas que tiene la red.

$$f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Figura 12. Función de pérdida.
Recuperado de:
https://learn.microsoft.com/es-es/archive/msdn-magazine/2019/april/artificiall
y-intelligent-how-do-neural-networks-learn

2.4.3.3 Gradiente descendiente

La gradiente descendiente es el gráfico de la función de costo que sirve de guía para ajustar las ponderaciones de los nodos para obtener un mejor resultado.

La pendiente de la curva hace de guía y señala el valor mínimo. El objetivo es encontrar dicho valor de toda la curva, que representa las entradas donde la red neuronal es más precisa.

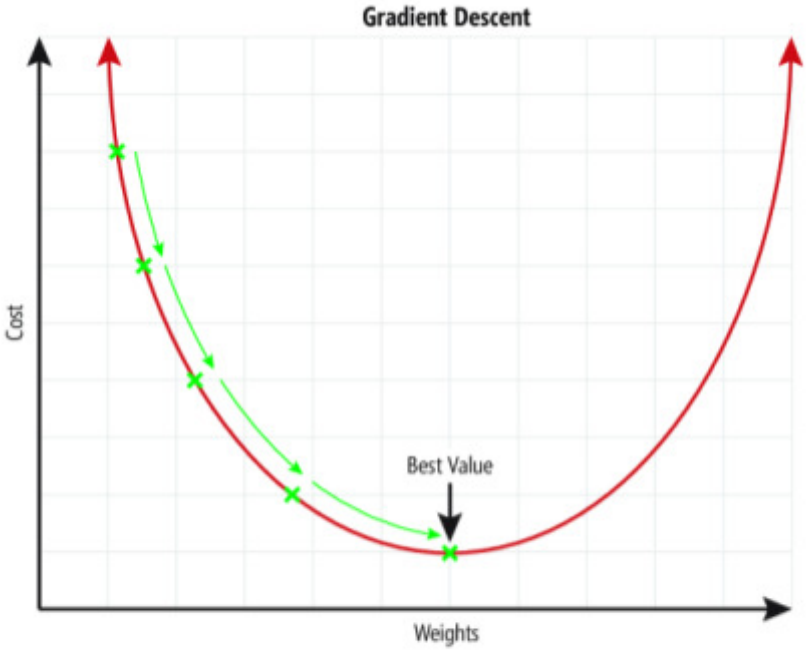

Figura 13. Gráfico de la gradiente Descendente. Recuperado de: https://learn.microsoft.com/es-es/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn

En la figura 13 se puede ver el gráfico de la gradiente donde cada suma de ponderaciones alcanza el punto más bajo, es decir, donde la pérdida será menor. Cuando la pendiente es negativa, se suman las ponderaciones y, en caso contrario, se restan. La cantidad de estás sumas y restas determina la velocidad de aprendizaje de la red. Si es demasiado grande, significa que ésta no podrá acertar en lo más mínimo; si es muy baja, tardará demasiado en aprender.

En la práctica, los gráficos de gradientes no son tan sencillos como el de la figura 13, sino que son más caóticos, observando subidas y bajadas por igual. El desafío es encontrar el punto más bajo en toda la curva. La mejor forma de hacer esto es tomando un punto aleatorio de la misma y avanzar por ella hacía el lado en el que el descenso sea más pronunciado.

2.4.3.4 Backward propagation

En está última instancia del proceso, se busca ajustar los pesos de cada neurona de tal manera que se minimice el error del resultado. Este algoritmo de *backpropagation* indica la responsabilidad que tiene cada neurona de que el resultado se haya desviado.

La forma en la que se calcula es desde la capa de salida hasta la de entrada, donde se realiza el cálculo y determina cuánta culpa tiene el resto de nodos. De ahí el nombre del algoritmo que significa *propagación hacia atrás.*

Para esto se realizan derivadas parciales en la función de costo con respecto a cada una de las variables, y así se minimiza la pérdida de la red, ajustando los sesgos y pesos de los nodos. Cuanto menor sea la pérdida, más precisa será la red para poder predecir resultados.

2.4.4 Underfitting

El *underfitting* (en español *subajuste,* fenómeno común en el aprendizaje automático) ocurre cuando un modelo es demasiado simple o es incapaz de capturar patrones en los datos de entrenamiento. El resultado de esto es que la red es incapaz de hacer predicciones precisas tanto en los datos de entrenamiento como en nuevos datos.

Algunas causas de esto pueden ser: un modelo demasiado simple, poca cantidad de datos de entrenamiento o que el dataset no tenga información suficiente para un problema. Esto se puede evitar aumentando la complejidad del modelo, así como un dataset con mayor cantidad de datos o mejores características.

Es importante encontrar un punto medio en la complejidad del modelo y en el conjunto de datos. Pero no es necesario que sea demasiado complejo porque también puede traer otros problemas como el *overfitting*.

2.4.5 Overfitting

El *overfitting* (en español *sobreajuste,* otro fenómeno común) ocurre durante el aprendizaje automático y es lo contrario al subajuste. En este caso, como se mencionó anteriormente, puede surgir por un modelo con demasiados datos y no puede generalizar los nuevos de modo correcto. Es decir, el modelo es demasiado complejo y no aprende los patrones generales y captura fluctuaciones aleatorias presentes en el dataset de entrenamiento.

3. Herramientas Utilizadas

A continuación se detallan las herramientas utilizadas en el desarrollo e investigación de la PPS.

3.1 Python

Python es uno de los lenguajes de programación multiplataforma de alto nivel más utilizados y,actualmente, con más crecimiento de usuarios. No sólo es un lenguaje orientado a objetos sino que también soporta la programación imperativa y funcional. Esto lo vuelve una herramienta muy versátil, se utiliza en todo tipo de aplicaciones, desde servicios de streamings hasta el desarrollo de aplicaciones de inteligencias artificiales.

Figura 14. Logo de python.
Recuperado de:
https://www.ironhack.com/es/es/blog/analisis-de-datos-con-python

Cuenta con una gran biblioteca de librerías que permiten que se pueda enfocar más en el desarrollo de aplicaciones lo cual explica su crecimiento en este campo.

La razón para utilizar este lenguajes, además de las características anteriormente descritas, es que el framework Tensor Flow y la librería Keras que son utilizados para el entrenamiento de las redes neuronales se pueden integrar con Python.

3.2 Google Colab

Figura 15. Logo de Google Colab.
Recuperado de:
https://www.marketing-branding.com/google-colaboratory-colab-guia-completa-espanol/

La herramienta Google Colab es un servicio en la nube que se basa en Jupyter y que permite que los usuarios lo puedan usar sin la necesidad de descargar el código. En otras palabras, permite que los usuarios desarrollen código en Python y utilicen sus librerías aprovechando la potencia que ofrece Google. Gracias a esto se convierte en una herramienta ideal para el desarrollo de inteligencias artificiales.

Cuenta con varias restricciones: los recursos de Colab no están garantizados ni son infinitos, los límites de uso a veces varían, no se pueden utilizar archivos multimedias que no estén conectados a la computación interactiva de Google Colab, entre otros. A pesar de esto, es sumamente práctica para poder aprender y desarrollar pilotos para Machine Learning y Deep Learning sin tener que invertir recursos locales como memorias y cpu.

3.3 TensorFlow

Figura 16. Logo de TensorFlow.
Recuperado de:
https://es.wikipedia.org/wiki/TensorFlow#/media/Archivo:TensorFlowLogo.svg

TensorFlow es un framework, una librería de código abierto para el aprendizaje automático desarrollado por el equipo de Google Brain. Es capaz de construir y entrenar modelos de aprendizaje automático y redes neuronales. Funciona mediante la definición y ejecución de grafos computacionales, gráficos dirigidos acíclicos que representan operaciones matemáticas y cálculos realizados por un modelo de aprendizaje automático.

En TensorFlow, los datos son representados como tensores (arreglos multidimensionales de valores numéricos) los cuales son manipulados mediante operaciones matemáticas, como multiplicación de matrices, sumas y funciones no lineales.

TensorFlow también proporciona herramientas para visualizar y monitorear el progreso del entrenamiento de la red, así como guardar y cargar modelos entrenados para su posterior uso.

En resumen, TensorFlow funciona mediante la definición y ejecución de grafos computacionales que representan operaciones matemáticas en los datos de entrada para construir y entrenar modelos de aprendizaje automático y redes neuronales.

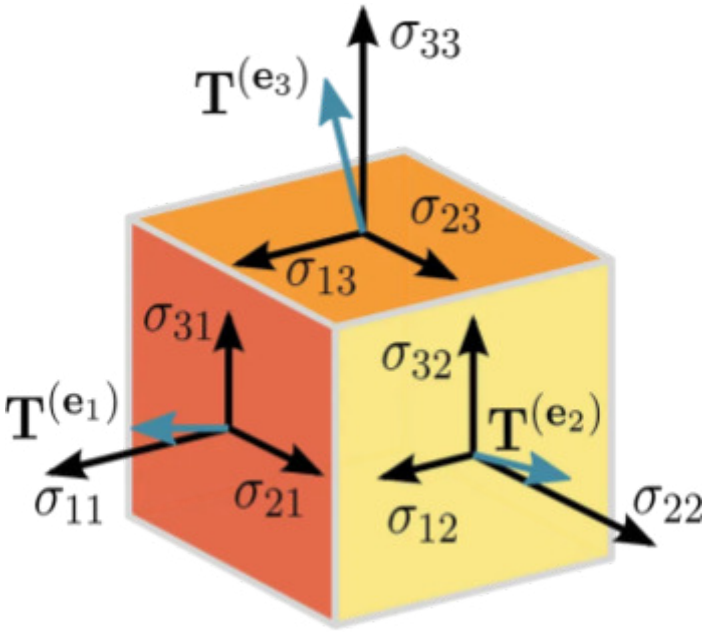

Figura 17. Ilustración de un tensor.
Recuperado de:
https://towardsdatascience.com/a-beginner-introduction-to-tensorflow-part-1-6d139e038278

3.4 Keras

Keras es una biblioteca de alto nivel para construir y entrenar modelos de aprendizaje automático en Python. Inicialmente fue desarrollada por François Chollet y se ha convertido en una de las bibliotecas más populares y utilizadas.

Ofrece una interfaz simple y fácil de usar para definir y entrenar modelos de Deep Learning. Permite a los usuarios definir capas de redes neuronales, conectarlas para formar modelos y entrenarlos con los datos que se requieran. También ofrece una amplia variedad de capas y funciones de activación predefinidas que se pueden usar para construir redes neuronales complejas.

Aunque su principal ventaja es la portabilidad, puede ejecutarse en diferentes plataformas y dispositivos, lo que hace posible implementar y desplegar modelos de manera fácil y rápida. Además, es compatible con varios Frameworks de Deep Learning como los es TensorFLow. Por estas razones será utilizada para el desarrollo de este proyecto.

3.5 Scikit-learn

Scikit-learn es una biblioteca de código abierto para Python que contiene una amplia gama de algoritmos de aprendizaje automático supervisado y no supervisado, así como herramientas para procesamiento de datos, evaluación de modelos, entre otros.

4. Desarrollo

4.1 Set de datos

Para poder hacer predicciones sobre los cultivos es necesario utilizar varios algoritmos que permitan el entrenamiento de una red neuronal que pueda predecir.

Una tarea fundamental a la hora de entrenar una red neuronal artificial es la preparación de un gran conjunto de datos que contengan miles de valores. La red debe conocer todos los elementos que se quieren predecir por lo que es necesario que los datos sean lo más fieles posibles a la realidad.

En este caso, se le presentará a la red un dataset que contiene distintos tipos de cultivos y condiciones óptimas para su crecimiento. Los distintos valores que incluye son los niveles de nitrógeno (N), fósforo (P) y potasio (K) de la tierra así como su pH, la temperatura (medida en grados celsius) y la humedad a la que están expuestos dichos cultivos, y, por último, la cantidad de mm de lluvia. A partir del entrenamiento de la red con estos datos será posible detectar anomalías o condiciones no óptimas para ciertos cultivos.

El dataset contiene datos correspondientes a distintas frutas, verduras, legumbres y plantas con miles de entradas y sus diversas condiciones en las que es posible sembrar los siguientes tipos de cultivos: *rice* (arroz), *maize* (maíz), *chickpea* (garbanzo), *kidney beans* (frijoles), *pigeon peas* (frijoles chícharos), *moth beans* (frijoles makusta), *mung bean* (frijol mungo), *black gram* (vigna mungo), *lentil* (lentejas), *pomegranate* (granada), banana, mango, *grapes* (uvas), *watermelon* (sandía), *muskmelon* (melón), *apple* (manzana), *orange* (naranja), papaya, *coconut* (coco), *cotton* (algodón), *jute* (yute) y *coffee* (café).[1]

[1] Los nombres se encuentran en inglés debido a que así es como son en el set de datos.

4.2 Análisis de dataset

A partir de este dataset se pueden sacar diferentes conclusiones de los diversos parámetros y valores que contienen. Por ejemplo, las distintas huertas deben tener diferentes parámetros para cada uno de los cultivos. A continuación se muestran los valores de potasio, fósforo y nitrógeno medios que debe tener la tierra para cada uno de los cultivos:

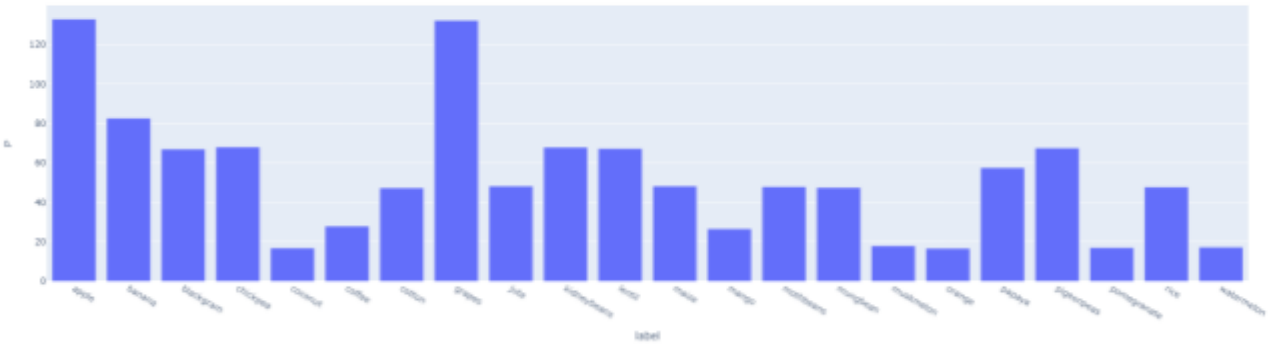

Figura 18. Representación de valores medios de fósforo para los distintos cultivos.
Fuente: Elaboración propia

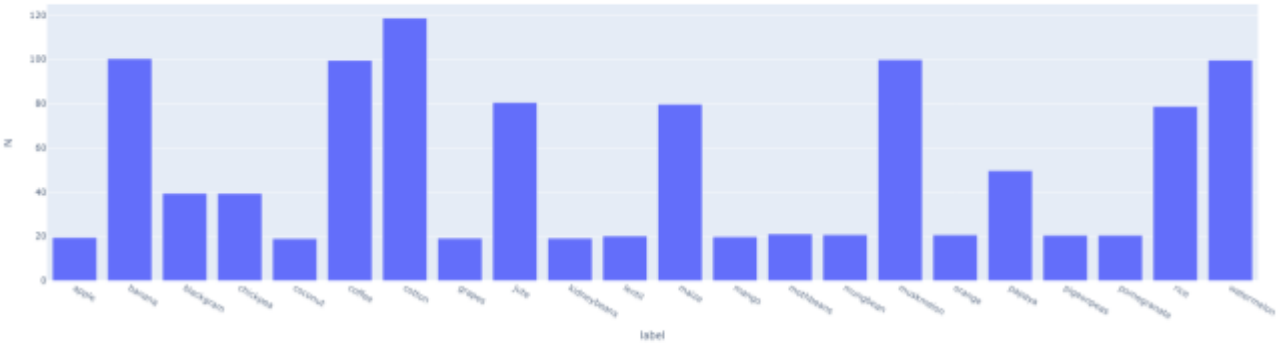

Figura 19. Representación de valores medios de nitrógeno para los distintos cultivos.
Fuente: Elaboración propia

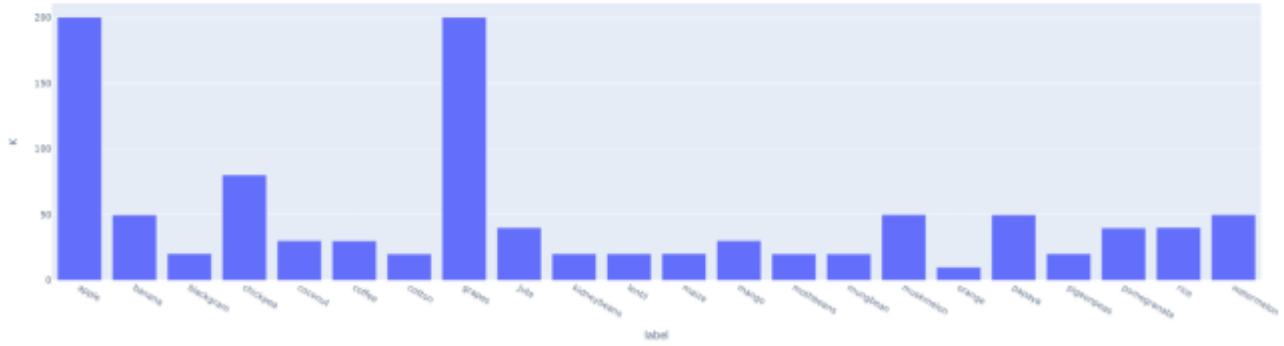

Figura 20. Representación de valores medios de potasio para los distintos cultivos.
Fuente: Elaboración propia

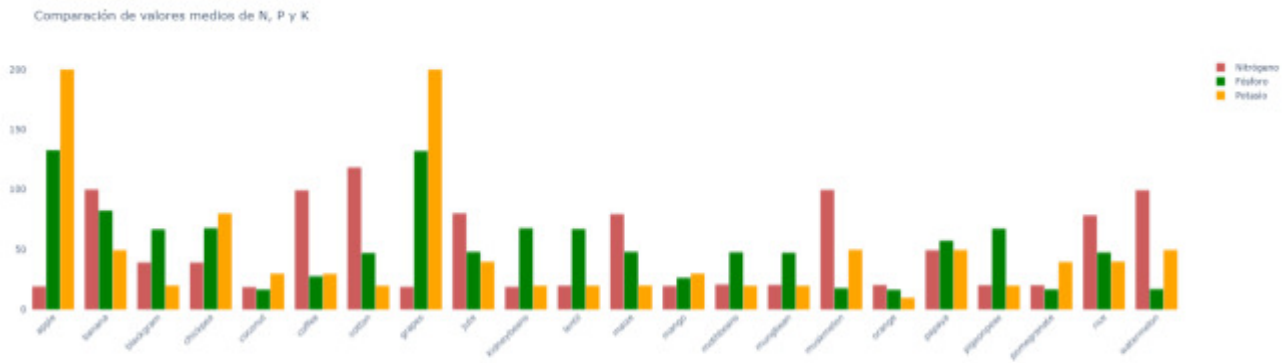

Figura 21. Comparación de N, P y K para los distintos cultivos.
Fuente: Elaboración propia

También se puede hacer un análisis sobre la relación entre la humedad y la temperatura a la que deben permanecer la huerta para sembrar de forma correcta cada una de las plantas, cómo se puede ver en los siguientes gráficos:

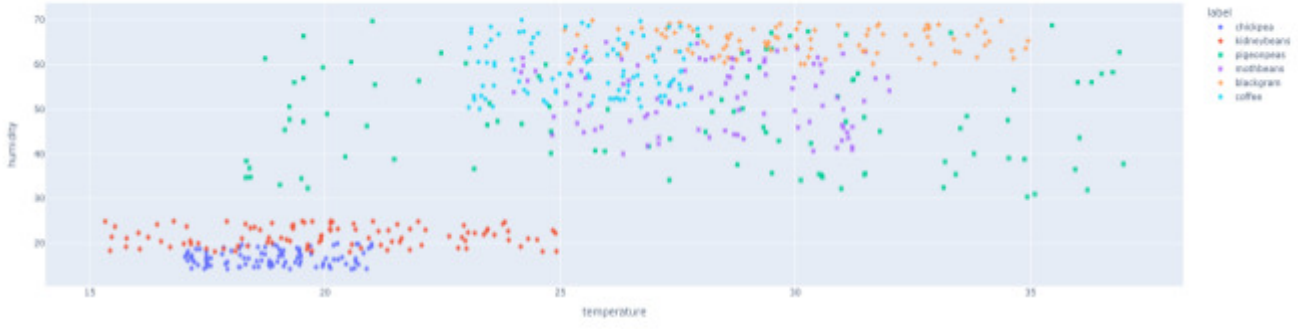

Figura 22. Distribución de datos en relación de humedad y temperatura para los cultivos: *chickpea, kidney beans, pigeon peas, moth beans, black gram* y *coffee*.
Fuente: Elaboración propia

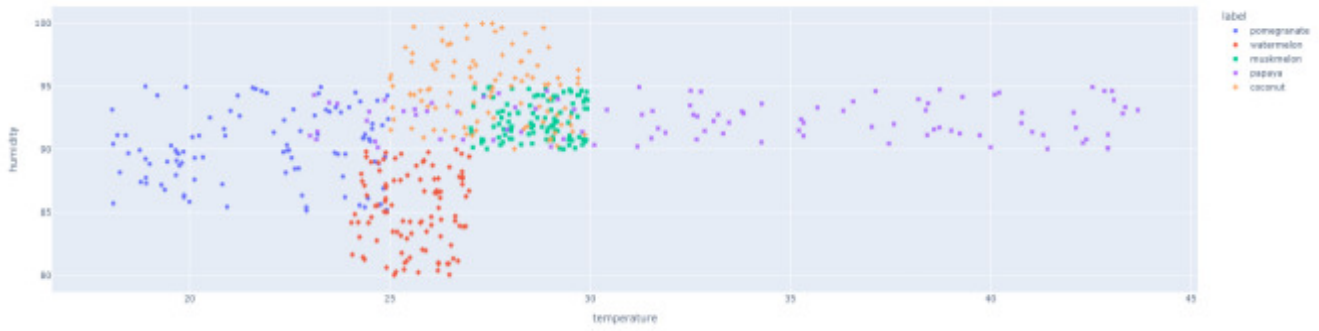

Figura 23. Distribución de datos en relación de humedad y temperatura para los cultivos: *pomegranate, watermelon, muskmelon, papaya* y *coconut*.
Fuente: Elaboración propia

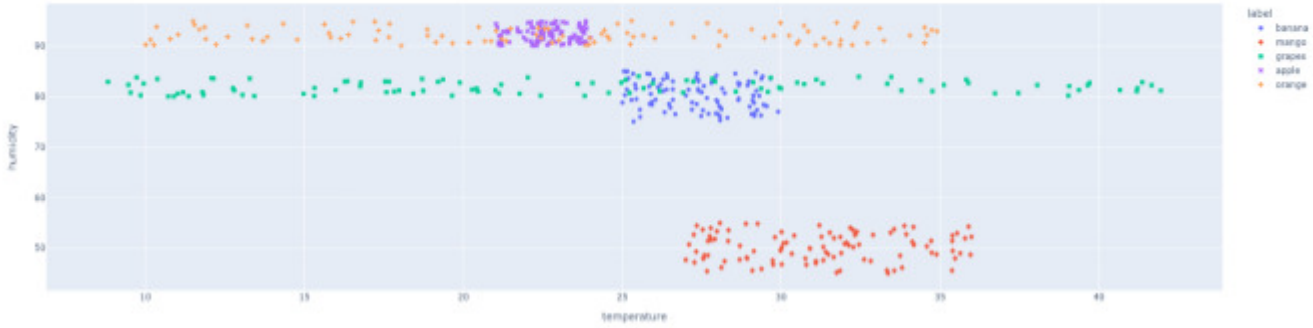

Figura 24. Distribución de datos en relación de humedad y temperatura para los cultivos: *banana, mango, oranges, apple* y *grapes*.
Fuente: Elaboración propia

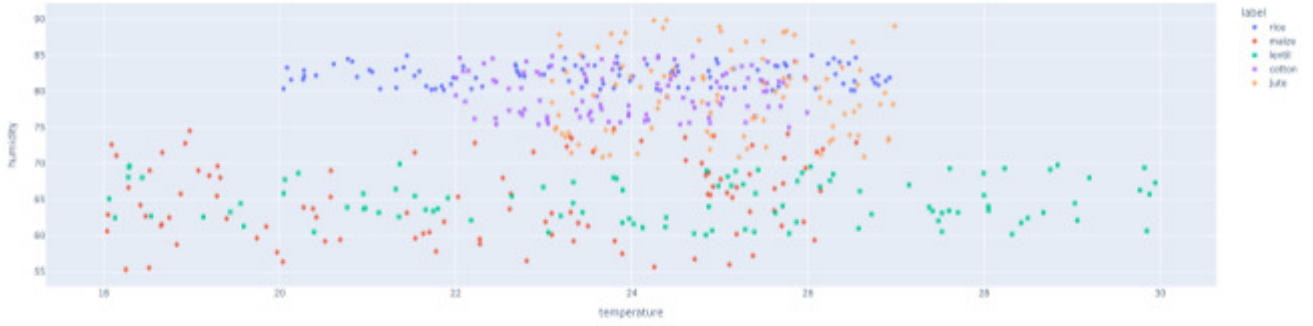

Figura 25. Distribución de datos en relación de humedad y temperatura para los cultivos: *rice, maize, lentil, cotton* y *jute*.
Fuente: Elaboración propia

Por último, se puede ver una matriz en la que se ve cómo es la correlación de las distintas características que tiene el set de datos para el aprendizaje de la red neuronal:

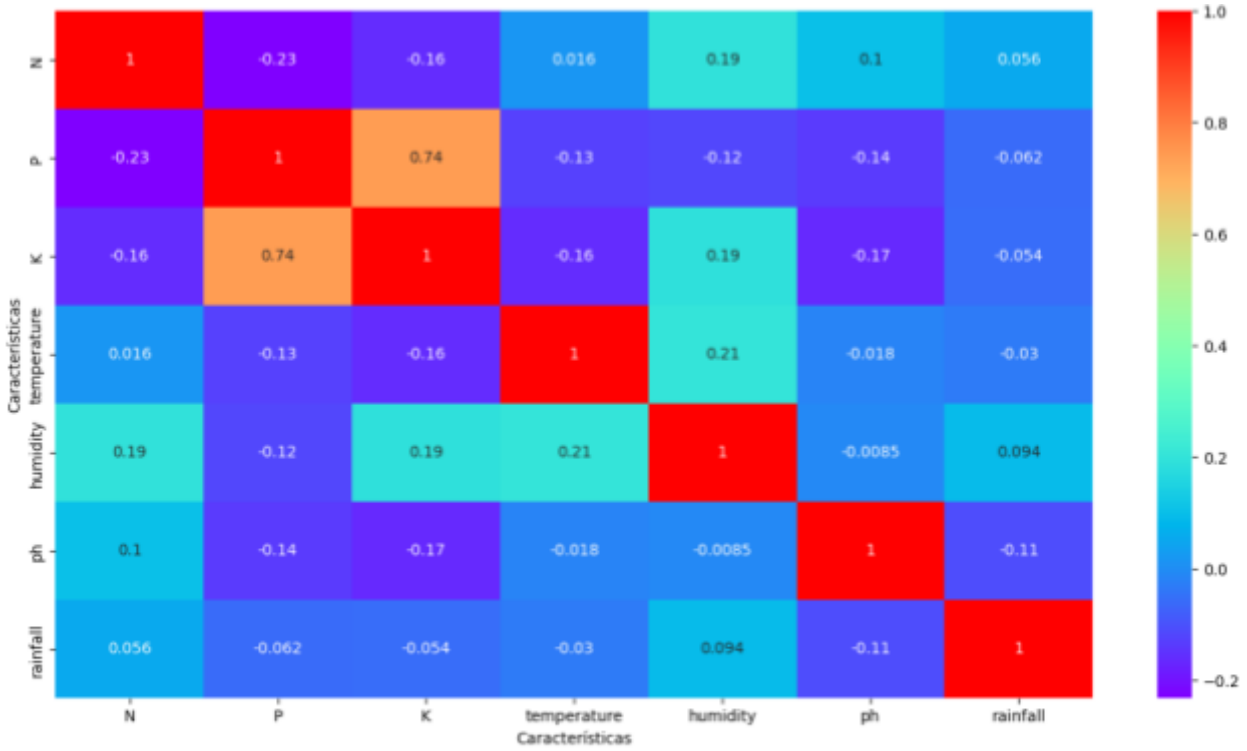

Figura 26. Correlación entre diferentes características.
Fuente: Elaboración propia

4.3 Predicción de tipo de cultivo

Con el dataset se puede crear un tipo de red neuronal que permita predecir el tipo de cultivo a partir de los valores introducidos manualmente por el usuario de nitrógeno, fósforo, etc. que haya en algún invernadero. La mayor utilidad de esto es poder elegir qué va a ser más eficiente plantar en una granja dependiendo de los distintos atributos de la misma.

Para lograr tener una red capaz de predecir estos resultados, es necesario emplear distintos algoritmos de aprendizaje que sean los más adecuados para el entrenamiento que requiere una red de clasificación. En otras palabras, se buscan algoritmos que ayuden a poder clasificar los resultados por etiquetas, en este caso siendo la etiqueta el nombre del cultivo.

Los diferentes algoritmos de aprendizaje, también conocidos como modelos, que se utilizaron para poder entrenar las redes neuronales de este tipo son: *Decision Tree* (DT), *Gaussian Naive Bayes* (NB), *Support Vector Machine* (SVM), *Random Forest* (RF) y *K-Nearest Neighbour* (KNN).

Primero hay que entrenar cada red utilizando estos algoritmos y después usar datos de pruebas para corroborar la exactitud de cada modelo. Para esto se emplea la librería *scikit-learn* para diferenciar los datos de prueba y de entrenamiento:

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(features,target,test_size = 0.2,random_state =2)
```

Figura 27. Creación de set de datos para entrenamiento y para evaluación de cada algoritmo.
Fuente: Elaboración propia

	N	P	K	temperature	humidity	ph	rainfall
5106	36	80	22	34.036195	64.287914	7.741419	66.855109
5847	80	25	55	29.540972	92.917783	6.163921	21.965308
677	13	54	15	28.660863	86.121946	6.860603	50.015343
2987	29	65	16	29.632105	65.913600	7.421608	71.163320
5433	109	71	53	28.654563	79.286937	5.695268	102.463378
...	...	..	..	...	...	...	...
2514	34	69	17	16.065228	18.724797	5.998125	88.066388
6443	81	46	45	23.255834	82.701593	7.124334	166.216085
3606	104	14	46	29.172209	92.214052	6.293486	21.302905
5704	90	11	50	26.587407	81.325632	6.932740	41.875400
2575	14	64	17	16.244692	21.357939	5.591704	66.970533

[5280 rows x 7 columns]

	N	P	K	temperature	humidity	ph	rainfall
5690	13	121	199	13.058097	80.282980	5.757010	70.756336
4000	15	6	35	26.762749	92.860569	6.420019	224.590366
4598	89	54	22	19.742133	59.662631	6.381202	65.508614
4719	3	69	18	23.042910	22.426110	5.833940	108.368432
3560	119	8	54	25.429009	82.914818	6.828983	56.341446
...	...	...	...	...	...	...	...
360	19	79	18	22.954582	24.035531	5.858618	107.731539
1261	31	136	203	23.819957	80.122116	6.002996	67.273986
622	14	58	19	29.650212	80.298683	6.489259	56.762784
5535	2	16	35	28.333333	51.395865	6.434198	91.672418
6169	34	49	45	35.676673	93.306419	6.586107	141.338117

[1320 rows x 7 columns]

Figura 28. Datos de entrenamiento (superior) y datos de prueba (interior).
Fuente: Elaboración propia

4.3.1 Decision Tree

Este algoritmo de aprendizaje supervisado es relativamente sencillo ya que, como su nombre lo indica, es un árbol de decisiones que comienza desde un nodo raíz. Las ramas salientes de éste alimentan a los nodos internos o nodos de decisión que realizan evaluaciones sobre los datos de entrada y así poder clasificarlos en los distintos subconjuntos que se indican mediante los nodos hoja, que representan todos los resultados posibles dentro del conjunto de datos.

Una desventaja que tiene este tipo de algoritmo es que es propenso al sobreajuste si no se controla la profundidad del árbol, además de que ya existen versiones mejores como el *random forest*.

Utilizando las herramientas que provee *scikit-learn* se puede calcular la precisión de cada algoritmo: el *recall* (que es la tasa de verdaderos positivos=, el *F1-score* (que es una métrica que se utiliza para evaluar el rendimiento de un modelo haciendo predicciones cuyo valor varía entre 0 y 1, siendo 1 un modelo perfecto); y, por último, el soporte (*support* en inglés, que son la cantidad de muestras de entrenamiento que pertenecen a cada etiqueta).

Una vez terminado el entrenamiento, se prueba qué tan preciso será a la hora de predecir un resultado utilizando los datos de prueba. Cómo se puede ver en la siguiente imágen, este modelo se puede ver que tiene una precisión del 92.12%:

Precisión de Decision Tree: 92.12121212121212

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	59
banana	1.00	1.00	1.00	54
blackgram	0.68	1.00	0.81	54
chickpea	1.00	1.00	1.00	65
coconut	1.00	1.00	1.00	58
coffee	1.00	0.90	0.95	59
cotton	1.00	1.00	1.00	71
grapes	1.00	1.00	1.00	53
jute	1.00	0.30	0.46	67
kidneybeans	1.00	0.80	0.89	65
lentil	0.88	1.00	0.94	59
maize	0.75	0.97	0.85	62
mango	1.00	0.93	0.96	56
mothbeans	1.00	0.47	0.64	60
mungbean	1.00	1.00	1.00	47
muskmelon	1.00	1.00	1.00	67
orange	1.00	1.00	1.00	63
papaya	1.00	1.00	1.00	59
pigeonpeas	0.94	1.00	0.97	64
pomegranate	1.00	1.00	1.00	60
rice	0.54	1.00	0.70	55
watermelon	1.00	1.00	1.00	63
accuracy			0.92	1320
macro avg	0.95	0.93	0.92	1320
weighted avg	0.95	0.92	0.91	1320

Figura 29. Resultados de prueba del algoritmo.
Fuente: Elaboración propia

4.3.2 Gaussian Naive Bayes

Este modelo de aprendizaje automático supervisado es utilizado para clasificar datos en diferentes categorías, asumiendo que los datos de entrada siguen una distribución gaussiana o distribución normal. El algoritmo utiliza los parámetros de esta distribución para poder calcular la posibilidad de que un dato pertenezca a una determinada clase. Es simple y rápido; se utiliza exclusivamente para clasificación ya que tiende a hacer suposiciones simplificadoras sobre la independencia de las características de los datos.

Al terminar el entrenamiento se puede ver que la precisión que tiene está red para predecir los datos de prueba es del 99,54%, como se puede ver en la siguiente imagen:

Precisión de GNB: 0.9954545454545455

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	59
banana	1.00	1.00	1.00	54
blackgram	1.00	1.00	1.00	54
chickpea	1.00	1.00	1.00	65
coconut	1.00	1.00	1.00	58
coffee	1.00	1.00	1.00	59
cotton	1.00	1.00	1.00	71
grapes	1.00	1.00	1.00	53
jute	0.96	0.99	0.97	67
kidneybeans	1.00	1.00	1.00	65
lentil	0.97	1.00	0.98	59
maize	1.00	1.00	1.00	62
mango	1.00	1.00	1.00	56
mothbeans	1.00	0.97	0.98	60
mungbean	1.00	1.00	1.00	47
muskmelon	1.00	1.00	1.00	67
orange	1.00	1.00	1.00	63
papaya	1.00	1.00	1.00	59
pigeonpeas	1.00	1.00	1.00	64
pomegranate	1.00	1.00	1.00	60
rice	0.98	0.95	0.96	55
watermelon	1.00	1.00	1.00	63
accuracy			1.00	1320
macro avg	1.00	1.00	1.00	1320
weighted avg	1.00	1.00	1.00	1320

Figura 30. Resultados de prueba del algoritmo.
Fuente: Elaboración propia

4.3.3 Support Vector Machine

El SVM es otro algoritmo de aprendizaje supervisado que se utiliza tanto para tareas de clasificación como de regresión. Para poder hacer predicciones se basa en el concepto de encontrar el hiperplano óptimo que separa los puntos de datos de diferentes clases de la manera más efectiva posible.

Es bastante eficiente, cómo se puede ver en la figura 31, tiene una exactitud del 99,01%, a cambio de requerir mayor costo computacional en comparación con el resto de los algoritmos.

Precisión de SVM: 0.9901515151515151

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	59
banana	1.00	1.00	1.00	54
blackgram	1.00	1.00	1.00	54
chickpea	1.00	1.00	1.00	65
coconut	1.00	1.00	1.00	58
coffee	1.00	1.00	1.00	59
cotton	1.00	1.00	1.00	71
grapes	1.00	1.00	1.00	53
jute	0.90	0.99	0.94	67
kidneybeans	1.00	1.00	1.00	65
lentil	0.94	1.00	0.97	59
maize	1.00	1.00	1.00	62
mango	1.00	1.00	1.00	56
mothbeans	1.00	0.93	0.97	60
mungbean	1.00	1.00	1.00	47
muskmelon	1.00	1.00	1.00	67
orange	1.00	1.00	1.00	63
papaya	1.00	0.98	0.99	59
pigeonpeas	1.00	1.00	1.00	64
pomegranate	1.00	1.00	1.00	60
rice	0.96	0.87	0.91	55
watermelon	1.00	1.00	1.00	63
accuracy			0.99	1320
macro avg	0.99	0.99	0.99	1320
weighted avg	0.99	0.99	0.99	1320

Figura 31. Resultados de prueba del algoritmo.
Fuente: Elaboración propia

4.3.4 Random Forest

El modelo RF es otro método de aprendizaje automático supervisado que, al igual que SVM, es utilizado tanto para clasificación como para regresión. En este caso, la idea es entrenarlo para tareas de clasificación, por lo tanto, para funcionar combina múltiples árboles de decisiones individuales, donde cada uno hace una predicción sobre los datos y la clase con más votos será la predicción final.

Es un algoritmo bastante robusto frente a valores atípicos y ruido en los datos, además de que puede manejar el sobreajuste y es efectivo para grandes cantidades de datos.

Demuestra ser más efectivo que los demás a la hora de hacer predicciones ya que como se puede ver en la figura 32 tiene una precisión del 99,69%.

Presición de Random Forest: 0.996969696969697

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	59
banana	1.00	1.00	1.00	54
blackgram	0.98	1.00	0.99	54
chickpea	1.00	1.00	1.00	65
coconut	1.00	1.00	1.00	58
coffee	1.00	1.00	1.00	59
cotton	1.00	1.00	1.00	71
grapes	1.00	1.00	1.00	53
jute	0.97	1.00	0.99	67
kidneybeans	1.00	1.00	1.00	65
lentil	0.98	1.00	0.99	59
maize	1.00	1.00	1.00	62
mango	1.00	1.00	1.00	56
mothbeans	1.00	0.97	0.98	60
mungbean	1.00	1.00	1.00	47
muskmelon	1.00	1.00	1.00	67
orange	1.00	1.00	1.00	63
papaya	1.00	1.00	1.00	59
pigeonpeas	1.00	1.00	1.00	64
pomegranate	1.00	1.00	1.00	60
rice	1.00	0.96	0.98	55
watermelon	1.00	1.00	1.00	63
accuracy			1.00	1320
macro avg	1.00	1.00	1.00	1320
weighted avg	1.00	1.00	1.00	1320

Figura 32. Resultados de prueba del algoritmo.
Fuente: Elaboración propia

4.3.5 K-Nearest Neighbour

El KNN es otro modelo de aprendizaje automático que, al igual que los anteriores, se utiliza tanto para regresión como para clasificación. Para poder clasificar, en primer lugar por cada valor ingresado se calculan los k vecinos más cercanos a este, posteriormente se realiza una votación y se determina a qué clase pertenece el nuevo valor.

Es un algoritmo fácil de implementar pero puede ser costoso a nivel computacional para grandes cantidades de datos ya que hay que calcular las distancias para todos los puntos en el conjunto de entrenamiento para cada predicción. A pesar de esto, tiene una gran tasa de acierto siendo de 98,33% como se ve en la siguiente figura:

Presición de KNN: 0.9833333333333333

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	59
banana	1.00	1.00	1.00	54
blackgram	0.98	1.00	0.99	54
chickpea	1.00	1.00	1.00	65
coconut	1.00	1.00	1.00	58
coffee	1.00	1.00	1.00	59
cotton	0.97	1.00	0.99	71
grapes	1.00	1.00	1.00	53
jute	0.90	0.99	0.94	67
kidneybeans	0.97	1.00	0.98	65
lentil	0.91	1.00	0.95	59
maize	1.00	0.97	0.98	62
mango	0.97	1.00	0.98	56
mothbeans	0.98	0.85	0.91	60
mungbean	1.00	1.00	1.00	47
muskmelon	1.00	1.00	1.00	67
orange	1.00	1.00	1.00	63
papaya	1.00	1.00	1.00	59
pigeonpeas	1.00	0.95	0.98	64
pomegranate	1.00	1.00	1.00	60
rice	0.98	0.87	0.92	55
watermelon	1.00	1.00	1.00	63
accuracy			0.98	1320
macro avg	0.98	0.98	0.98	1320
weighted avg	0.98	0.98	0.98	1320

Figura 33. Resultados de prueba del algoritmo.
Fuente: Elaboración propia

4.3.6 Detección de momento óptimo de cultivo

Para finalizar con está sección, se desarrolló un algoritmo con el único propósito de poder determinar los mejores meses para plantar los diferentes cultivos según lo requiera el usuario.

El cálculo que permite determinar el mejor mes es una función sencilla que consiste de una colección de datos, que se componen de dos partes: el cultivo y los mejores meses correspondientes al mismo.

Para hacer una predicción, se introduce el nombre de la planta que se desea y la función retorna como resultado e indica la mejor época para plantar.

4.4 Detección de valores atípicos

No es suficiente solo con poder predecir qué tipo de cultivo será el más adecuado para ciertas condiciones ya que las condiciones pueden ir variando con el paso del tiempo. Por eso es necesario poder detectar si hay valores fuera de lo común en las peticiones, que puedan estar afectando el desarrollo de las plantas.

Para esto se investigaron algoritmos capaces de poder detectar valores atípicos en conjuntos de datos numéricos. La idea es que se puedan ir viendo atributo por atributo a ver si hay algún valor que esté muy por encima o muy por debajo de la media y así, poder corregirlo para un crecimiento óptimo de los cultivos.

Al igual que con los algoritmos anteriores, es necesario dividir los datos de prueba y los datos de entrenamiento para poder calibrar las redes neuronales. El algoritmo que se utilizó para el entrenamiento fue el siguiente: *Elliptic Envelope*.

A pesar de utilizar el mismo set de datos que las redes neuronales para la detección de tipo de cultivos, es necesario dividir al set en varios subconjuntos por cada planta, es decir, uno para el arroz, otro para las manzanas, etc. Esto se debe a que los algoritmos utilizados al entrenarse con todos los datos juntos pueden tener errores a la hora de predecir qué datos son atípicos, ya que no puede distinguir cuales son de una planta y cuales de otra.

4.4.1 Elliptic Envelope Algorithm

Este método es una técnica de detección de anomalías en un conjunto de datos. Se basa en el concepto de “envolvente elíptica”, que es una forma geométrica que rodea los datos normales y excluye las observaciones anómalas.

Este algoritmo asume que los datos normales siguen una distribución gaussiana multivariante, es decir, se distribuyen de manera similar a una elipse en un espacio multidimensional. La idea sería ir ajustando la elipse alrededor de los datos normales y, luego, considerar los datos que se encuentran fuera de la elipse como anomalías potenciales, cómo se ve en la siguiente figura donde los datos atípicos se remarcan con un color rojo:

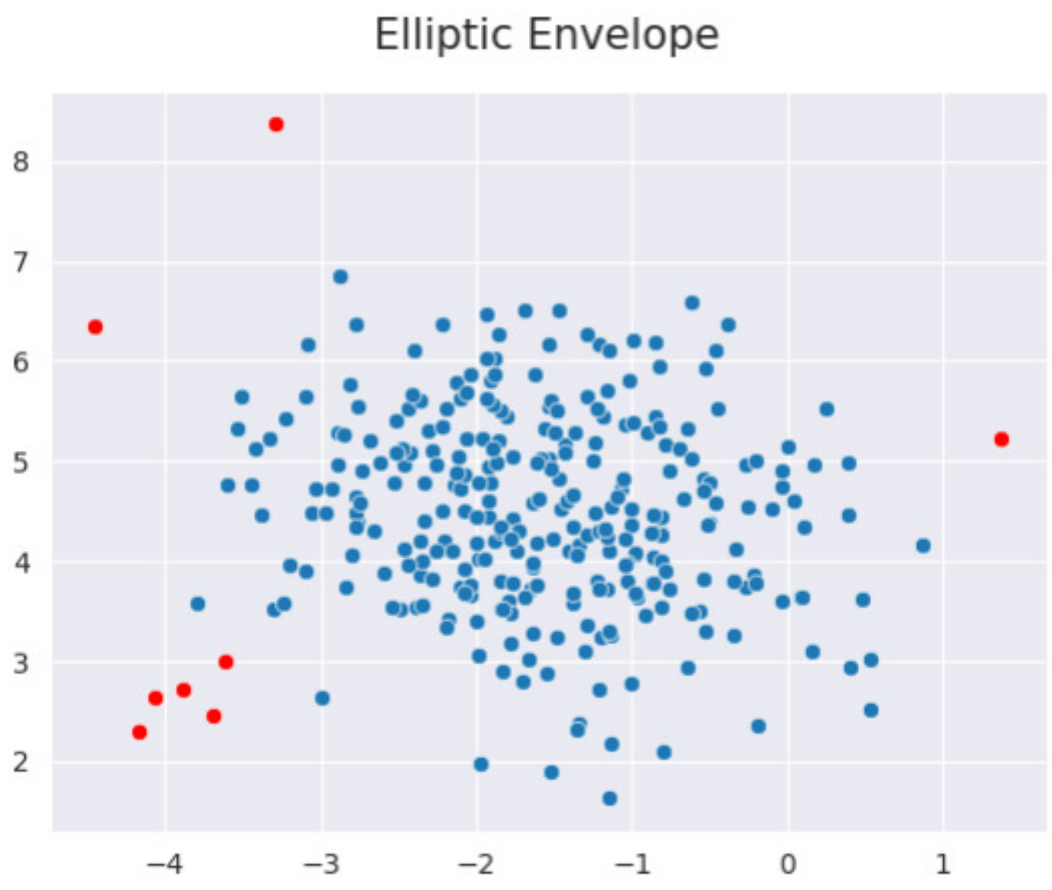

Figura 34. Gráfico de ejemplo de Elliptic Envelope..
Fuente: Elaboración propia

A pesar de que el algoritmo funciona y no es muy difícil de implementar, es necesario que los datos normales se distribuyen de forma gaussiana y que los datos se alejen significativamente de la distribución. De lo contrario, el rendimiento se puede ver afectado y, en algunos casos, puede ser necesario ajustar algunos parámetros del algoritmo o directamente considerar técnicas más avanzadas.

La idea, al igual que con la predicción de tipo de cultivo, es entrenar el algoritmo usando datos de prueba para después poder hacer una predicción sobre los datos reales.

5. Análisis de resultados

5.1 Predicción de cultivos

Una vez finalizado el entrenamiento con los diferentes algoritmos para está tarea, se pueden realizar las predicciones con los diferentes modelos ingresando valores de N, K, P, T°, humedad, Ph y lluvia.

Para poner a prueba los distintos algoritmos se utilizaron los mismos datos para ver en qué categoría son clasificados, en las siguientes imágenes se puede ver las diferentes predicciones que tiene cada algoritmo usando los mismos datos:

- Decision Tree:

```
data = np.array([[6,49,42,20,90,6.5,150]])
prediction = DecisionTree.predict(data)
print(prediction)
```

```
['papaya']
```

Figura 35. Predicción de Decision Tree.
Fuente: Elaboración propia

- Gaussian Naive Bayes:

```
data = np.array([[6,49,42,20,90,6.5,150]])
prediction = NaiveBayes.predict(data)
print(prediction)
```

```
['papaya']
```

Figura 36. Predicción de GNB.
Fuente: Elaboración propia

- SVM:

```
data = np.array([[6,49,42,20,90,6.5,150]])
prediction = SVM.predict(data)
print(prediction)
```

```
['mothbeans']
```

Figura 37. Predicción de SVM.
Fuente: Elaboración propia

- Random Forest:

```
data = np.array([[6,49,42,20,90,6.5,150]])
prediction = RF.predict(data)
print(prediction)
```

```
['papaya']
```

Figura 38. Predicción de Random Forest.
Fuente: Elaboración propia

- KNN:

```
data = np.array([[6,49,42,20,90,6.5,150]]
prediction = knn.predict(data)
print(prediction)
```

```
['rice']
```

Figura 39. Predicción de KNN.
Fuente: Elaboración propia

Si bien la mayoría de los modelos parecen coincidir en que la categoría correcta es la papaya, los algoritmos KNN y SVM predicen otros resultados siendo estos *rice* y *mothbeans* respectivamente.

En este caso, lo correcto sería elegir la categoría que tenga mayoría o en todo caso tener como opción más probable al algoritmo con mayor exactitud de todos, es decir, *random forest*, cómo se puede ver en el siguiente cuadro comparativo de los distintos modelos.

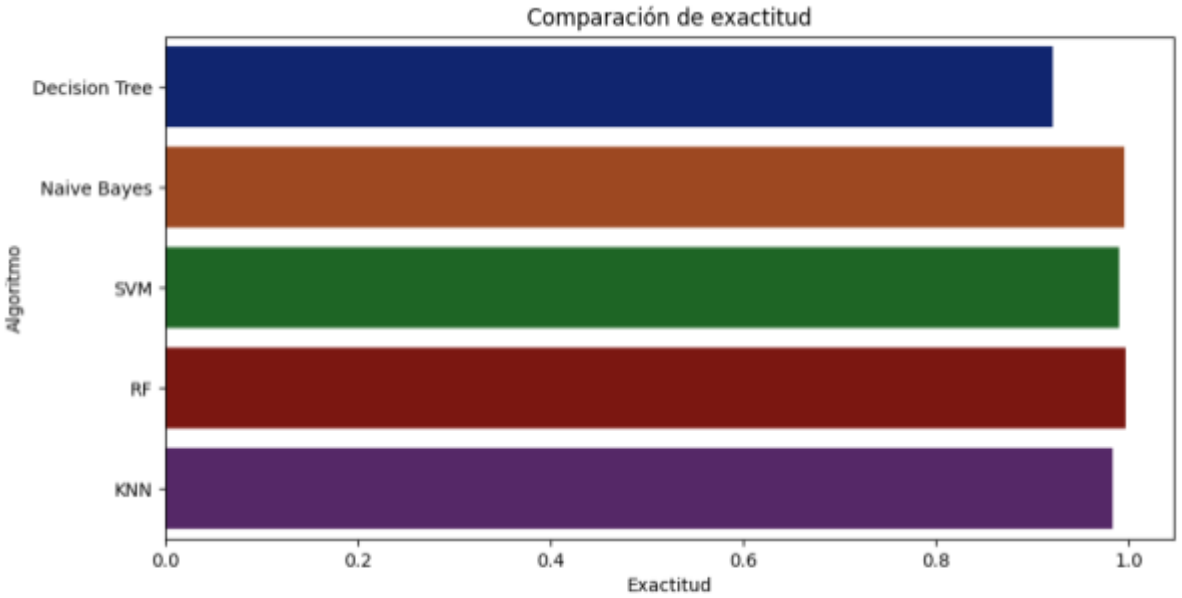

Figura 40. Comparación de exactitud de los modelos implementados.
Fuente: Elaboración propia

Estas predicciones son altamente útiles a la hora de elegir qué tipo de cultivo plantar. Una vez implementada la huerta se pueden tomar mediciones de las condiciones a las qué está y así poder cultivar las plantas que se adapten mejor a ese ambiente. Incluso, también es útil para saber qué parámetros ajustar en cierta huerta que permitan obtener las mejores condiciones posibles para poder sembrar.

Ahora ya es posible determinar qué cultivo plantar en cierta huerta pero ahora mismo el algoritmo no indica cuándo sería la mejor época para plantar. Para solucionar este problema, se diseñó un algoritmo que procesa el resultado de la predicción anterior y así puede determinar cuales son los mejores meses para plantarlo.

Por ejemplo, suponiendo que la predicción sobre una huerta específica es que la mejor opción para plantar son manzanas, ésta es procesada por la nueva función e indica los mejores meses para plantar de la siguiente manera:

```
apple should be plant in: june july
```

Figura 41. Predicción de los mejores meses para plantar manzanas.
Fuente: Elaboración propia

5.2 Detección de valores atípicos

Una vez terminado el entrenamiento de está red neuronal, se pueden realizar predicciones sobre qué datos en un conjunto son anormales. Para hacer esto es necesario aplicar el método sobre un conjunto de datos nuevo, para poder identificarlos se agrega un nuevo atributo a cada muestra llamado *outlier* (valor atípico) que puede tener dos posibles valores: 1 en caso de que los valores de N, K, P, T°, humedad, Ph y lluvia de la muestra están dentro de los parámetros, o -1 en caso de que alguno de los valores sea irregular, como se en las siguiente imágen:

	N	P	K	temperature	humidity	ph	rainfall	outliers
83	75	59	40	26.043720	84.969072	5.999969	186.753677	1
30	86	37	38	21.446540	84.943760	5.824709	272.201720	1
56	70	50	43	25.655535	83.470211	7.120273	217.378858	1
24	97	45	36	23.483813	81.332651	7.375483	224.058116	1
16	69	53	38	21.587118	82.788371	6.249051	276.655246	1
23	82	43	45	21.052536	82.678395	6.254028	233.107582	-1
2	98	56	37	23.004459	82.320763	7.840207	263.964248	1
27	75	51	37	24.529227	80.544986	7.070960	260.263403	1
28	92	60	41	20.775761	84.497744	6.244841	240.081065	-1
13	100	56	35	24.014976	82.056872	6.984354	185.277339	1
99	82	51	38	23.359054	83.595123	5.333323	188.413665	1
92	86	35	45	26.528728	80.122675	6.158377	218.916357	1
76	92	44	41	26.730724	81.785968	7.868475	280.404439	1
14	86	42	35	25.665852	80.663850	6.948020	209.586971	1
0	77	49	42	20.879744	82.002744	6.502985	202.935536	1
21	69	60	36	25.157455	83.117135	5.070176	231.384316	1
3	78	41	42	26.491096	80.158363	6.980401	242.864034	1
29	73	59	40	22.301574	80.644165	6.043305	197.979121	1
61	62	51	41	25.755286	83.518271	5.875346	245.662680	1
79	68	46	42	22.678461	83.728744	7.524080	200.913316	-1

Figura 42. Predicción de valores atípicos.
Fuente: Elaboración propia

Como se muestra en la figura 42, hay sólo tres datos que se destacan por tener valores que se alejan demasiado de la media. Lamentablemente, la función no puede indicar qué valor específico es atípico y queda en manos del usuario poder determinar eso.

Otra desventaja encontrada usando este algoritmo es que hay que tener los datos para entrenamiento separados por cada cultivo. En otras palabras, para entrenar esta red neuronal con este algoritmo es necesario tener varios archivos y correr el proceso por cada uno.

6. Conclusiones

Durante el desarrollo e implementación de este trabajo se estudiaron diversos algoritmos de Machine Learning para la clasificación de distintos tipos de cultivos. Luego, para la detección de valores atípicos con el objetivo de determinar las características óptimas del ambiente para la selección del cultivo que permita maximizar el rendimiento de cada tipo al momento de obtener sus frutos. Cabe destacar que también se realizó una investigación sobre la teoría de las redes neuronales y los diversos métodos y tipos de redes que hay. Finalmente, se plantearon los resultados obtenidos y en esta sección se presentan algunas conclusiones y posibles mejoras que pueda haber en un futuro.

Se buscó una forma de determinar las mejores condiciones para el correcto desarrollo de las plantas para su cultivo y poder determinar el momento óptimo para hacerlo. De está forma, poder ayudar a los agricultores al darles herramientas para tener un mejor control sobre sus trabajo y tener más información disponible a la hora de determinar las mejores condiciones para el desarrollo de cada cultivo, o corregir errores antes de que su cosecha se vea afectada por alguna circunstancia adversa.

Otro punto para destacar fue la elección del Framework Tensorflow y el lenguajes de programación Python para trabajar con ML. ya que tienen una gran cantidad de funciones y librerías que facilitan mucho el trabajo. Así como también lo cómodo que es poder trabajar en Google Collab que tiene una gran capacidad de procesamiento, ahorrando mucho tiempo para el entrenamiento, que en una computadora personal llevaría mucho tiempo.

En cuanto a los resultados obtenidos para el set de datos de pruebas se puede decir que fueron correctos. Lamentablemente no se pudieron realizar pruebas en ambiente reales aunque sí se cumplieron con las expectativas del trabajo. Se puede decir que el campo de la inteligencia artificial, especialmente el de Machine Learning, será la base para los

proyectos del futuro debido a su gran uso sin importar el campo en el que se utilicen.

6.1 Posibles mejoras y líneas futuras

En primer lugar, se puede mejorar y aumentar el set de datos para poder hacer un re-entrenamiento de las redes neuronales para que sean más exactas. Además poder crear y utilizar set de datos con plantas locales de la República Argentina en vez de tener que depender de datos del exterior que no reflejan completamente ni nuestros cultivos ni nuestras condiciones, a lo largo y ancho del país.

En segundo lugar, perfeccionar la detección de valores atípicos para que pueda indicar qué valores son los que se destacan por sobre los demás, y así corregir los errores con mayor facilidad y evitar tener que sobre analizar los datos.

Por último, una gran forma de probar la red sería realizar pruebas sobre ambientes de trabajo reales haciendo una integración con diversos sensores que puedan monitorear los cultivos para probar la efectividad del proyecto.

Reflexión sobre la PPS como espacio de formación:

La realización de este trabajo sirvió para poder introducirme en el mundo de la inteligencia artificial y el Machine Learning, campos que cada vez más se utilizan en ámbitos profesionales y para proyectos realmente ambiciosos. Todo esto es muy importante para mi carrera a nivel profesional. Si bien no es un tema que se haga mucho hincapié a lo largo de la formación académica, me ayudó a comprender mejor varios conocimientos adquiridos a lo largo de la carrera.

Seleccioné este proyecto debido a la temática que trataba, a pesar de no tener experiencia previa sobre el desarrollo en este tipo de tecnologías. De esta forma pude tener un acercamiento más interesante del que puede hacer algún curso o investigación sin ningún tipo de objetivo más que el de satisfacer la curiosidad de saber cómo funcionan las redes neuronales.

A nivel personal, considero está última instancia formativa como una oportunidad de incorporar nuevos conocimientos sobre un tema que viene en crecimiento en el mundo de la informática como lo es la IA, y también el momento de volcar lo aprendido durante la carrera para poder hacerlo de la mejor forma.

7. Bibliografía

Agritutorials, 2022 - Classsification of Crops Based on 3 Growing Seasons - https://agritutorials.com/classification-of-crops-rabi-kharif-zaid/

Amazon, 2023 - Aprendizaje por refuerzo en AWS DeepRacer - https://docs.aws.amazon.com/es_es/deepracer/latest/developerguide/deepracer-how-it-works-overview-reinforcement-learning.html

Amazon, 2023 - ¿Qué es una red neuronal? - https://aws.amazon.com/es/what-is/neural-network/

Amazon, 2023 - ¿Qué es Python? - https://aws.amazon.com/es/what-is/python/

Aurélien, Géron (2019) *Hands on Machine Learning with Scikit-learn, Keras & TensorFlow*, editorial O'reilly, Estados Unidos.

Calvo Diego, 2018 - Backpropagation - Redes neuronales - https://www.diegocalvo.es/backpropagation-redes-neuronales/

Calvo Diego, 2017 - Definición de una red neuronal artificial - https://www.diegocalvo.es/definicion-de-red-neuronal/

Chen James, 2023 - What is a Neural Network - https://www.investopedia.com/terms/n/neuralnetwork.asp

Douglas da Silva, 2021 - Diferencia entre Machine Learning y Deep Learning - https://www.zendesk.com.mx/blog/machine-learning-deep-learning-diferencias/

Estefanía Freire, Sarahí Silva, 2019 - Redes neuronales - https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

Frank La La, 2019 - Inteligencia Artificial: ¿Cómo aprenden las redes neuronales? - https://learn.microsoft.com/es-es/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn

François, Chollet (2017) *Deep Learning With Python*, editorial Manning, Estados Unidos.

Gandhi Rohith, 2018 - Support Vector Machine – Introduction to Machine Learning Algorithms - https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Guarav Pahuja, 2021 - Machine Learinng - All you need to know about Outliers - https://www.linkedin.com/pulse/machine-learning-all-you-need-know-outliers-gaurav-pahuja/

Gupta Aryan, 2023 - An introduction to Scikit-Learn: Machine Learining in Python - https://www.simplilearn.com/tutorials/python-tutorial/scikit-learn

Heller Martin, 2019 - What is Keras? - https://keras.io/about/

IBM, 2023 - ¿Qué es KNN? - https://www.ibm.com/mx-es/topics/knn

IBM, 2023 - What are Neural Networks? - https://www.ibm.com/topics/neural-networks

IBM, 2023 - What is underfitting? - https://www.ibm.com/topics/underfitting

IBM, 2023 - What is overfitting? - https://www.ibm.com/topics/overfitting

IBM, 2023 - What is Random Forest? - https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems.

Irene Alvarado, 2023 - Redes neuronales - https://ml4a.github.io/ml4a/es/neural_networks/#:~:text=A%20menudo%20al%20t%C3%A9rmino%20b,bajo%20lo%20hace%20m%C3%A1s%20f%C3%A1cil.

Keras, 2023 - About Keras - https://keras.io/about/

Narasimha Prasanna HN, 2017 - A begginer introduction to TensorFlow (Part 1) - https://towardsdatascience.com/a-beginner-introduction-to-tensorflow-part-1-6d139e038278

Oscar García-Olalla Olivera, 2019 - Redes Neuronales artificiales: Qué son y cómo se entrenan - https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i#:~:text=Entrenar%20una%20red%20neuronal%20consiste,a%20los%20datos%20que%20conocemos.

Pragati Baheti, 2023 - Activation Functions in Neural Networks [12 Types & Use Cases] - https://www.v7labs.com/blog/neural-networks-activation-functions#:~:text=The%20linear%20activation%20function%2C%20also,the%20value%20it%20was%20given.

Prashant Gupta, 2017 - Decision Trees in Machine Learning - https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Rubiales Alberto, 2020 - ¿Qué es Underfitting y Overfitting? - https://rubialesalberto.medium.com/qu%C3%A9-es-underfitting-y-overfitting-c73d51ffd3f9

Rukshan Pramoditha, 2021 - 4 Machine Learning techniques fot outlier detection in Python - https://towardsdatascience.com/4-machine-learning-techniques-for-outlier-detection-in-python-21e9cfacb81d

Vats Rohan, 2021 - Gaussian Naive Bayes: What you need to know? - https://www.upgrad.com/blog/gaussian-naive-bayes/

Yegulalp Serdar, 2022 - What is TensorFlow? The Machine Learning library explained - https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html