UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENT DE MATEMÀTICA APLICADA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Matemática Aplicada

Problemas de agrupamiento en ganadería de precisión

Trabajo Fin de Máster

Máster Universitario en Investigación Matemática

AUTOR/A: Sanjuan Silvestre, Sergi

Tutor/a: Calabuig Rodriguez, Jose Manuel

Cotutor/a: García Raffi, Luis Miguel

Director/a Experimental: ARNAU NOTARI, ANDRES ROGER

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE MATEMÁTICA APLICADA

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

TRABAJO DE FIN DE MÁSTER

PROBLEMAS DE CLASIFICACIÓN Y PREDICCIÓN EN GANADERÍA DE PRECISIÓN

Tutores: Lluís Miquel Garcia Raffi,
Jose Manuel Calabuig Rodriguez y
Andres Roger Arnau Notari

Sergi Sanjuan Silvestre

InvestMat

Curso académico 2022-23

Resumen

La ganadería de precisión es una técnica moderna que utiliza tecnología avanzada, como sensores, dispositivos de seguimiento y análisis de datos, para monitorear y administrar de manera eficiente la cría y el manejo de animales de granja. Este trabajo consiste en una aplicación de esta técnica a terneras de ganadería no extensiva. Para esto, se ha hecho uso de modelos matemáticos de inteligencia artificial para maximizar la eficiencia, la salud y el bienestar de las terneras pertenecientes a este tipo de granjas, donde no se cuenta con muchos animales.

Este estudio cuenta con el respaldo de dos proyectos de gran magnitud: Re-livestock y TED-Farm, cuyos objetivos se centran en mejorar la gestión y el rendimiento de las operaciones ganaderas, al tiempo que se promueve la salud y el bienestar animal. Estos proyectos nos han proporcionado valiosos datos sobre las actividades de cada animal en un tiempo determinado, así como el grupo alimenticio de cada animal: terneras de 2 *tomas, de* 3 *tomas o destetadas. De este modo, el objetivo principal de este estudio es clasificar estas terneras por su grupo alimenticio mediante sus actividades diarias, el cual ha proporcionado resultados prometedores. Como segundo y último objetivo, se pretende predecir las actividades que realizará una ternera perteneciente a un determinado grupo.*

Para lograr estos objetivos, se emplean múltiples métodos y algoritmos de aprendizaje automático. Estos métodos se detallan a lo largo del documento, y se proporciona un apéndice al final que ofrece una explicación más detallada de estos métodos, incluyendo pruebas matemáticas y herramientas utilizadas en el estudio.

Palabras clave: *Ganadería de Precisión, ganadería no extensiva, Inteligencia Artificial, Aprendizaje Automático, Redes Neuronales, análisis de series temporales.*

Abstract

Precision livestock farming is a modern technique that utilizes advanced technology such as sensors, tracking devices, and data analysis to efficiently monitor and manage the breeding and handling of farm animals. This work involves the application of this technique to intensively raised calves. To achieve this, mathematical models of artificial intelligence have been used to maximize the efficiency, health, and welfare of calves in these types of farms with a limited number of animals.

This study is supported by two large-scale projects: Re-livestock and TED-Farm, which aim to improve the management and performance of livestock operations while promoting animal health and welfare. These projects have provided valuable data on the activities of each animal over a specific time period, as well as the feeding group of each animal: calves with 2 feedings, calves with 3 feedings, or weaned calves. Thus, the main objective of this study is to classify these calves into their respective feeding groups based on their daily activities, which has yielded promising results. As a second and final objective, the intention is to predict the activities that a calf belonging to a specific group will perform.

To achieve these objectives, multiple methods and machine learning algorithms are employed. These methods are detailed throughout the document, and an appendix is provided at the end, offering a more comprehensive explanation of these methods, including mathematical proofs and tools used in the study.

Key words: *Precision Livestock Farming, non-extensive farming, Artificial Intelligence, Machine Learning, Neural Networks, time series analysis.*

Índice

Índice de figuras

Índice de cuadros

Prefacio

En un mundo en constante evolución, impulsado por avances tecnológicos sin precedentes, la Inteligencia Artificial (IA) se ha convertido en un faro de esperanza y una fuerza transformadora. El campo del aprendizaje automático (machine learning) ha emergido como el epicentro de esta revolución, desencadenando un sinfín de posibilidades y oportunidades sin precedentes. Hoy en día, nos encontramos en el umbral de una nueva era, donde la IA está impulsando cambios fundamentales en todas las esferas de nuestra sociedad.

El machine learning, en su esencia, es la disciplina que permite a las máquinas aprender de manera autónoma a través de la experiencia y mejorar su rendimiento a medida que se enfrentan a nuevos datos. Es el proceso que permite a las computadoras identificar patrones ocultos, comprender relaciones complejas y generar conocimiento útil a partir de enormes conjuntos de datos. Esta rama de la inteligencia artificial ha revolucionado nuestra capacidad para analizar información a gran escala y ha transformado la forma en que abordamos problemas y tomamos decisiones.

Dentro del emocionante campo del machine learning, las redes neuronales se destacan como una de las herramientas más poderosas y versátiles. Inspiradas por la estructura y el funcionamiento del cerebro humano, estas redes pueden aprender y adaptarse a patrones complejos en los datos, lo que las convierte en una herramienta invaluable para abordar problemas difíciles y desafiantes.

Imaginemos cómo estas tecnologías están revolucionando la atención médica, permitiendo diagnósticos más precisos, descubriendo nuevos tratamientos y mejorando la calidad de vida de las personas. Pensemos en cómo las redes neuronales están revolucionando el sector financiero, detectando fraudes de manera más efectiva, optimizando carteras de inversión y revolucionando el comercio electrónico al ofrecer recomendaciones personalizadas y experiencias de compra sin problemas.

Más allá de los beneficios tangibles, la inteligencia artificial también nos desafía a repensar y reinventar nuestro enfoque hacia el trabajo y la creatividad. Esta tecnología nos brinda una oportunidad única de liberar nuestra capacidad creativa y dejar que las máquinas se encarguen.

1. Introducción

En la actualidad, la **Ganadería de Precisión** (PLF, de sus siglas en inglés "*Precision Livestock Farming*") ha generado un cambio significativo en la forma en que se crían y gestionan los animales en las granjas. Se trata de una metodología que utiliza tecnologías avanzadas, como sensores, sistemas de información y análisis de datos, para optimizar la gestión y el rendimiento de las actividades ganaderas, es decir, para lograr una producción ganadera más eficiente, sostenible y rentable. La unión de la ganadería de precisión con la **inteligencia artificial** ha abierto un amplio abanico de posibilidades para mejorar el bienestar animal, aumentar la eficiencia de producción y optimizar la toma de decisiones informadas en este sector.

La ganadería de precisión se ha convertido en una metodología clave en el manejo de animales de **granjas no extensivas**. Estas granjas, también conocidas como granjas intensivas, son sistemas de producción ganadera que se caracterizan por tener una alta densidad de animales o cultivos en un espacio limitado. A través del uso de tecnologías avanzadas y sistemas de monitoreo, se recopilan datos en tiempo real sobre los animales. Sensores y dispositivos de seguimiento instalados en los animales permiten obtener mediciones precisas de parámetros como la temperatura corporal, los movimientos, el consumo de alimento y agua, entre otros factores que influyen en la salud y el bienestar de los animales. Otro tipo de sensores que desempeñan un papel crucial en la ganadería de precisión son los ambientales, ya que permiten recopilar datos sobre el entorno y las condiciones ambientales en las que se encuentra el ganado, como la temperatura ambiental, la humedad, la calidad de aire o los niveles de ruido. La recopilación de estos datos permite a los ganaderos tomar decisiones más informadas sobre la gestión del ganado, ajustar las condiciones del entorno para mejorar la salud y el rendimiento de los animales, prevenir enfermedades y optimizar los recursos utilizados en la producción ganadera.

En este contexto, la inteligencia artificial desempeña un papel fundamental en el análisis e interpretación de los datos recopilados. Mediante el uso de algoritmos de aprendizaje automático, es posible procesar grandes volúmenes de información y extraer conocimientos valiosos. Estos algoritmos tienen la capacidad de identificar patrones, correlaciones y relaciones en los datos, proporcionando información clave para la toma de decisiones y la mejora del manejo de los animales.

En esta situación, dos **proyectos**, "*Re-livestock*" y "*TED-Farm*", están interesados en la aplicación de tecnologías en la ganadería no extensiva. Ambos proyectos han recopilado una gran cantidad de datos sobre los animales en las granjas, que servirán como base para

crear modelos y obtener resultados relevantes.

Se describen a continuación los dos proyectos principales de este trabajo:

- **Re-livestock – "Facilitating Innovations for Resilient Livestock Farming Systems"**, con referencia 101059609 (Página web del proyecto).
  (*Re-livestock – "Facilitando las innovaciones para sistemas de producción ganadera resilientes"*).

  Este es el único proyecto concedido en la convocatoria de Horizonte Europa HORIZON-CL6-2021-CLIMATE-01-06, con duración 2022-2027 y un presupuesto de 12 millones de euros. Con la coordinación del CSIC y la participación de 37 socios, la UPV (formada por 6 investigadores entre los que se encuentran los directores de este trabajo) es líder del WP4 destinado a evaluar estrategias de mitigación y adaptación de las granjas al cambio climático.

  Este proyecto tiene como **objetivo general** comprender y promover la adopción de prácticas innovadoras aplicadas a diferentes escalas (animal, rebaño/granja, sector y región) para reducir las emisiones de gases de efecto invernadero (GEI) en la ganadería y aumentar la capacidad de hacer frente a los impactos del cambio climático, con el fin último de aumentar la resiliencia general del sector ganadero.

  Dentro de este marco, existen **objetivos particulares** en el proyecto que están directamente relacionados con este trabajo, pues podría considerarse una posible continuación del mismo. Algunos de estos objetivos son:

  1. **Refinar y aplicar herramientas tecnológicas innovadoras de evaluación ambiental y socioeconómica a escala agrícola para mejorar la adopción de prácticas**. En este sentido, se busca utilizar tecnologías avanzadas para evaluar de manera precisa y completa el impacto ambiental y socioeconómico de las prácticas ganaderas. Esto permitirá obtener datos fiables y relevantes que respalden la toma de decisiones informadas en relación con la adopción de prácticas sostenibles.
  2. **Desarrollo de algoritmos matemáticos para tratar la información generada por sensores en las granjas**. En este objetivo se busca procesar y analizar los datos recopilados con el fin de obtener información útil y tomar decisiones informadas en la gestión de la granja. Estos algoritmos permiten extraer conocimientos y patrones a partir de los datos brutos, lo que ayuda a

los agricultores y ganaderos a optimizar sus operaciones, mejorar la eficiencia y maximizar los resultados.

Estos objetivos particulares del proyecto ofrecen una oportunidad relevante para continuar este trabajo actual y contribuir al avance de la investigación y la aplicación de soluciones innovadoras en el ámbito de la ganadería sostenible. La combinación de herramientas tecnológicas avanzadas, evaluaciones ambientales y socioeconómicas, así como el diseño de estrategias de transición, permitirá impulsar la adopción de prácticas más sostenibles y resilientes en el sector ganadero, contribuyendo así a la mitigación del cambio climático y al fortalecimiento de la capacidad de adaptación frente a sus impactos.

- **TED-Farm – "Facilitating ecological transition of livestock production through the digitalisation of farming systems"**, con referencia TED2021-130759B-C31.
(*TED-Farm – "Facilitando la transición ecológica de la producción ganadera mediante la digitalización de los sistemas de producción"*).

  Este es un proyecto, con duración 2022-2024, financiado por la convocatoria de Proyectos de Transición Ecológica y Digital 2021 de la Agencia Estatal de Investigación y coordinado por UPV (*subproyecto* 1), entre los que se encuentran los directores de este trabajo, con la participación de Neiker (*subproyecto* 2) y CSIC-INIA (*subproyecto* 3).

  Este proyecto tiene como objetivo utilizar los datos digitales como herramienta para la toma de decisiones con el fin de transformar la producción ganadera en sistemas más sostenibles y resilientes. Sin embargo, para aprovechar esta oportunidad única, es necesario abordar la brecha de conocimiento entre los datos generados y el potencial uso de la información que contienen. Esto implica utilizar la información recopilada en la granja para múltiples propósitos en un enfoque integrado. Este conocimiento es fundamental para facilitar la transición hacia una producción ganadera más sostenible basada en la gestión digital, incluso para sistemas ganaderos extensivos y de bajo consumo de recursos.

  El **objetivo general** de este proyecto es facilitar el uso de los datos digitales generados en las granjas para cuantificar y mejorar la capacidad de mitigación y adaptación de la producción animal. Para lograr esto, se requiere un trabajo multidisciplinario que involucre la combinación de herramientas estadísticas avanzadas con disciplinas de producción animal como genética, nutrición, bienestar animal, estructuras

y medio ambiente. Al integrar estas áreas de conocimiento, se busca optimizar el manejo de los animales, mejorar su rendimiento productivo y reducir los impactos ambientales asociados a la ganadería.

Dentro de este proyecto, el *subproyecto 1*, coordinado por la UPV, tiene como **objetivo particular** facilitar la transición de los ganaderos hacia un sector ganadero digitalizado en el que la gestión de datos juega un papel clave en la transición ecológica. Las actividades para lograr este objetivo particular incluye definir todos los aspectos relacionados con la adquisición, almacenamiento, tratamiento y uso de datos, así como las herramientas utilizadas para el análisis de acuerdo con los objetivos de mitigación y adaptación. El proyecto recopilará los datos de granjas comerciales en los que estos se generan de forma rutinaria. Estos datos serán tratados siguiendo un enfoque coordinado a lo largo del proyecto. Dentro de este objetivo, el proyecto evaluará y promoverá la viabilidad del uso de datos digitales con fines de mitigación y adaptación, desde la selección e instalación de sensores hasta la gestión y procesamiento de datos y la toma de decisiones. Teniendo esto en cuenta, se pueden observar en la Figura 1 las tareas correspondientes a este subproyecto.

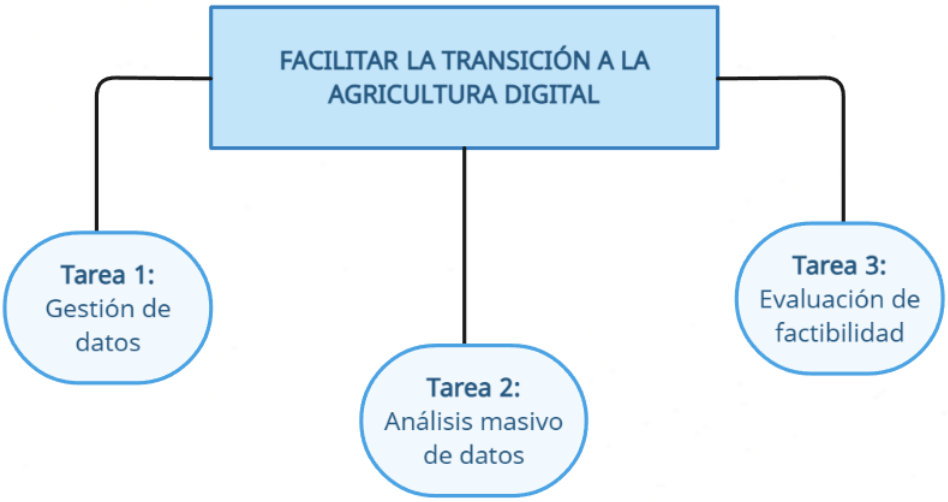


Figura 1: Tareas del *subproyecto* 1 (UPV).

Estos proyectos serán el enfoque principal de este trabajo, dado que han sido los que nos han proporcionado los datos. En el marco de estos proyectos, se han llevado a cabo una recopilación exhaustiva de datos acerca de sus terneras, que se utilizarán como base

para la implementación de los diferentes métodos de inteligencia artificial de este trabajo. Estos datos incluyen información detallada sobre la situación de cada ternero en un tiempo determinado, así como el grupo al que pertenecen, es decir, si son terneras de 2 tomas, de 3 tomas o destetadas (explicado en la Sección 5).

La disponibilidad de estos datos proporciona una valiosa oportunidad para realizar análisis y predicciones en la ganadería no extensiva. Por ejemplo, mediante el uso de algoritmos de machine learning, en particular mediante redes neuronales, a lo largo de la Sección 6 veremos si es posible predecir el grupo al que pertenece cada cría de ternero a partir de sus movimientos y comport amientos registrados. Esta capacidad de predicción permite optimizar la alimentación y las prácticas de cuidado, adaptándolas a las necesidades específicas de cada grupo y mejorando el crecimiento y la salud de estos animales.

Además de la predicción del grupo, los datos recopilados también pueden ser utilizados para intentar predecir los próximos movimientos que realizará un ternero en un tiempo determinado. Veremos que existen tipos de redes neuronales, al igual que algoritmos de machine learning tradicional, que analizan los datos históricos de movimientos y comportamiento de dichos animales, y a partir de ellos, pueden inferir los movimientos futuros. Esta información es valiosa para planificar el manejo del espacio, diseñar instalaciones adecuadas y gestionar el bienestar animal de manera óptima.

Para lograr estos objetivos, a lo largo de este trabajo se va a hacer uso de múltiples métodos matemáticos. En la Sección 4 se explican todos estos métodos matemáticos necesarios para la comprensión completa del trabajo. En primer lugar se muestran todos los métodos y algoritmos, tanto de machine learning tradicional como de redes neuronales artificiales, utilizados para la construcción del clasificador. Por último, se explican las redes neuronales recurrentes, utilizadas en el predictor. Si se desea una explicación más detallada acerca de estos métodos, donde se presentan pruebas matemáticas y algún que otro ejemplo, se recomienda ir al Apéndice A.

Destacar que todo el desarrollo de este trabajo se ha realizado mediante *Python*, un lenguaje de programación que cuenta con una gran cantidad de librerías especializadas en el machine learning, como son el caso de *TensorFlow*, *Keras* y *scikit-learn*, utilizadas en este trabajo. En el Apéndice B se puede encontrar una explicación acerca de las librerias utilizadas en el trabajo, y los motivos del porqué se ha elegido cada una.

En conclusión, en este trabajo se pretende aclarar como la aplicación de la inteligencia artificial en la ganadería no extensiva, en particular en la cría de terneros, puede revolucionar la forma en que se crían y gestionan estos animales en las granjas. Mediante la combinación de la ganadería de precisión y la inteligencia artificial, se aprovechan al má-

ximo los datos recopilados, proporcionando información valiosa para la toma de decisiones informadas y la optimización del manejo de los terneros. Con el respaldo de dos grandes proyectos vigentes dedicados a esta área, se están sentando las bases para una ganadería no extensiva más avanzada y sostenible, que aproveche todo el potencial de la inteligencia artificial en beneficio de los animales, los agricultores y la sociedad en general.

Nota: Los archivos de este proyecto son accesibles para descarga en el repositorio de GitHub del autor y que se puede acceder mediante [15].

El código puede ser descargado y hacer uso del mismo, excepto los datos, que en nuestro caso no son públicos. Al tratarse de un código focalizado a los experimentos, la estructura del mismo ha sugerido una división por cada modelo implementado.

2. Interés del proyecto

2.1. Los retos de la ganadería

La **producción ganadera** se encuentra en un momento crucial en el cual debe enfrentar el desafío de seguir produciendo alimentos seguros y de calidad de manera rentable, reduciendo su impacto ambiental, mejorando el bienestar animal y manteniendo su relación con la sociedad. Además, destaca que el contexto internacional presenta inestabilidades en la producción y suministro de materias primas para la alimentación animal.

En los últimos años, se han logrado avances significativos en mejora genética, alimentación e instalaciones, lo que ha resultado en mejoras productivas en las granjas. Sin embargo, el crecimiento actual de la ganadería sigue apuntando a trayectorias insostenibles debido a la magnitud del sector, por lo que se requiere una mayor ambición en términos de eficiencia y cambio de paradigmas de producción.

La Unión Europea está reconsiderando su modelo productivo y se enfoca en la transformación ecológica y digital en los próximos años, con el objetivo de reducir el uso de recursos, minimizar emisiones y residuos, y mejorar el bienestar animal y la seguridad alimentaria.

En lo referente al bienestar animal, sigue siendo un desafío la aplicación de métodos no invasivos para su monitoreo, con énfasis en la capacidad de actuación en tiempo real y en el seguimiento del bienestar a lo largo de la vida del animal.

La alimentación animal representa un coste económico importante y tiene un impacto ambiental significativo, especialmente en la producción intensiva de monogástricos. Se destaca la necesidad de comprender el comportamiento de los animales en relación con la ingesta y utilización de alimentos para lograr un uso más preciso de los mismos.

Aunque los temas abordados anteriormente corresponden a campos de conocimiento diversos, se subraya la estrecha relación entre ellos y la necesidad de un enfoque integrado para superar los desafíos mencionados.

2.2. Situación de la ganadería de precisión

La **ganadería de precisión** es una herramienta con gran potencial para abordar de manera integral los desafíos de la producción ganadera. Su objetivo es gestionar individualmente a los animales a través de la monitorización continua de su salud, bienestar, parámetros productivos e impacto ambiental. Esto permite un enfoque más personalizado

y la toma de decisiones en tiempo real.

Ya existen aplicaciones prácticas de esta técnica en el mercado, pero aún existen desafíos científico-técnicos para analizar los procesos y tomar decisiones prospectivas. En este trabajo se abordará uno de los desafíos principales, la implementación de técnicas de análisis masivo de información en aspectos específicos de la producción animal.

Inicialmente desarrollada para vacas lecheras, la ganadería de precisión se adapta a otras especies como cerdos o terneros en nuestro caso, adaptando los métodos de monitorización a cada especie. Además, es importante optimizar el uso de sensores en las granjas y promover un uso integral de la información con fines productivos, de bienestar y ambientales.

2.3. Oportunidades que ofrece la ganadería de precisión

La ganadería de precisión ofrece una oportunidad única para integrar aspectos productivos, ambientales, nutricionales y de bienestar animal que han sido abordados de manera separada en el pasado. Aunque persisten desafíos tecnológicos en el desarrollo de sensores y su conectividad con los sistemas de recopilación de datos, se han logrado avances significativos. Además, existen métodos para el análisis masivo de la información generada por la ganadería de precisión y el equipo de investigación cuenta con especialistas en este campo.

Este trabajo/proyecto tiene como objetivo explorar un aspecto poco estudiado en la literatura: comprender al animal como un ente cuyo comportamiento impacta en su productividad, bienestar, consumo de recursos, emisiones contaminantes y calidad de los productos. Se aplicarán las herramientas existentes, como sensores y algoritmos, a la información generada en la granja o en proyectos de investigación en curso, con el fin de resolver aspectos prácticos y agregar valor a estas tecnologías.

La ganadería de precisión se considera no como un fin en sí misma, sino como un medio para mejorar el conocimiento del funcionamiento del animal y su aplicación en todos los aspectos mencionados anteriormente relacionados con la producción ganadera.

2.4. Aportaciones de la ganadería de precisión a la ganadería no extensiva

Enfocándonos en la ganadería no extensiva, que constituye la base de este estudio, la ganadería de precisión presenta las siguientes contribuciones a este tipo de sistema

ganadero, el cual se caracteriza por tener un número reducido de animales.

1. **Monitoreo individualizado de cada animal** para un análisis más detallado y personalizado de su salud, comportamiento y rendimiento.
2. **Optimización precisa de la alimentación y nutrición de cada animal**, considerando sus necesidades específicas, lo que resulta en un mejor crecimiento y salud.
3. **Control y ajuste personalizado del entorno**, garantizando condiciones óptimas de bienestar y reduciendo el estrés animal.
4. **Seguimiento exhaustivo de la reproducción y salud reproductiva de cada animal**, permitiendo una gestión más efectiva de la reproducción y una mejora genética más precisa.
5. **Utilización de datos individuales para una toma de decisiones informada y personalizada**, que contribuye a la eficiencia general del sistema ganadero y a la maximización del rendimiento de cada animal.

En resumen, la diferencia de la aplicación de este enfoque tecnológico a la ganadería no extensiva frente a la extensiva es que en este caso se pueden hacer análisis individualizados y personalizados para cada animal.

3. Objetivo del trabajo

El **objetivo principal** de este proyecto es investigar la posibilidad de clasificar las terneras en diferentes grupos según su alimentación: grupos en los que toman 2 tomas, 3 tomas o terneras destetadas, utilizando únicamente las actividades que realizan a lo largo de un día. Esto plantea la necesidad de desarrollar un **modelo de clasificación**. En caso de obtener resultados prometedores en esta clasificación, como **segundo objetivo** se pretende también predecir las actividades que realizará una ternera perteneciente a un grupo específico.

Para lograr esto, en primer lugar, se realizará la predicción del grupo al que pertenece cada animal. Posteriormente, utilizando la información de las actividades que ha llevado a cabo en un intervalo de tiempo determinado, se buscará predecir las actividades que la ternera llevará a cabo en momentos posteriores, ya sea en intervalos de una hora, medio día, un día completo o incluso períodos más largos. Esto nos conduce a emplear un **análisis de series temporales con datos discretos**.

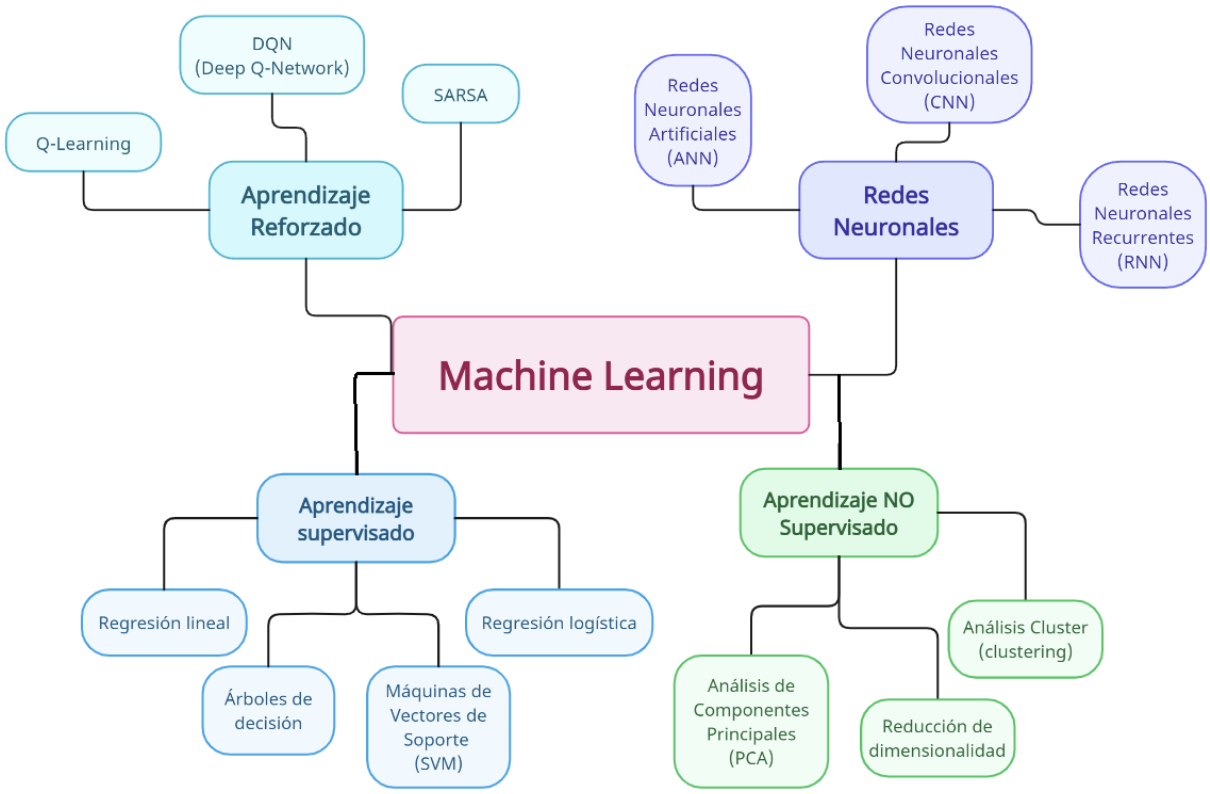
Este proyecto se basa en la premisa de que las actividades realizadas por las terneras pueden proporcionar indicios sobre su grupo de alimentación y también pueden servir como información para predecir sus actividades futuras. La clasificación precisa de las terneras y la capacidad de predecir sus actividades futuras podrían tener aplicaciones importantes en la gestión del cuidado y la alimentación de las terneras en la industria ganadera.

Para lograr estos objetivos, en este proyecto se van a emplear distintos **métodos matemáticos**, tanto de machine learning tradicional para la clasificación en grupos, como de redes neuronales para el análisis de series temporales. En la siguiente sección se van a explicar todos estos métodos y algoritmos para una completa comprensión del trabajo. Si se desea una explicación más detallada acerca de estos métodos, donde se presentan pruebas matemáticas y algún que otro ejemplo, se recomienda ir al Apéndice A.

4. Modelos matemáticos

4.1. Introducción al Machine Learning

El **Aprendizaje Automático** (ML, de sus siglas en inglés "*Machine Learning*") es un campo de estudio dentro de la **Inteligencia Artificial** que se centra en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender y mejorar automáticamente a partir de los datos sin ser programadas explícitamente. En lugar de seguir instrucciones específicas, los algoritmos de aprendizaje automático analizan patrones y estructuras en los datos para realizar tareas o tomar decisiones.



En este esquema se pueden observar algunos de los enfoques, métodos y algoritmos que caracterizan el machine learning. Dentro de este, se puede diferenciar el **aprendizaje automático clásico** y el **moderno** en que el moderno incluye las redes neuronales y métodos de mayor complejidad.

El ML clásico está caracterizado por los siguientes enfoques: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje reforzado. La diferencia básica entre estos radica en la presencia o ausencia de etiquetas en los datos de entrenamiento.

En el **aprendizaje supervisado**, se proporcionan ejemplos de datos de entrenamiento con etiquetas o salidas deseadas. El algoritmo aprende a realizar una asignación entre las características de entrada y las etiquetas correspondientes, lo que le permite predecir o clasificar nuevos ejemplos en función de los ejemplos de entrenamiento con etiquetas proporcionados. En este ámbito, los **problemas de clasificación** son ampliamente reconocidos y se encuentran entre los más frecuentes. Estos problemas implican trabajar con conjuntos de datos que están previamente etiquetados y organizados en diferentes categorías. Nuestro objetivo principal es poder asignar correctamente una categoría específica a un nuevo dato que no ha sido previamente etiquetado. En otras palabras, deseamos clasificar adecuadamente el nuevo dato dentro de las categorías conocidas en base a los patrones y características presentes en los datos etiquetados.

En el **aprendizaje no supervisado**, no se proporcionan etiquetas en los datos de entrenamiento. El algoritmo busca descubrir patrones o estructuras inherentes en los datos sin ninguna guía explícita. El objetivo es encontrar agrupamientos, relaciones o características ocultas en los datos sin conocimiento previo de las categorías específicas. En este enfoque de aprendizaje, los **problemas de agrupamiento** son ampliamente reconocidos y se encuentran entre los más comunes. En estos problemas, nos encontramos frente a un conjunto de datos sin etiquetar y nuestro objetivo es descubrir posibles relaciones o estructuras subyacentes en ellos. Nos interesa determinar si estos datos pueden agruparse en categorías o conjuntos distintos en base a los valores de las variables observadas.

El **Apredizaje Reforzado** (RL, de sus siglas en inglés "*Reinforcement learning*") es un enfoque del aprendizaje automático que se basa en la interacción de un agente con un entorno para aprender a tomar decisiones óptimas. A través de un **proceso de ensayo y error**, el agente aprende a través de la retroalimentación recibida del entorno, que puede ser positiva (recompensa) o negativa (castigo). Este tipo de aprendizaje se inspira en la psicología conductual y se basa en el concepto de refuerzo, donde el agente busca maximizar una recompensa acumulada a largo plazo. Es importante mencionar que los problemas más comúnmente asociados con el aprendizaje reforzado son aquellos que involucran la toma de decisiones secuenciales en un entorno dinámico y no determinista, como la planificación de rutas, el control de sistemas y la robótica, entre otros.

Figura 2: Aprendizaje Reforzado.

En proyectos que involucran machine learning, incluyendo este trabajo, se sigue un enfoque común que consta de una serie de **pasos fundamentales** (desarrollados en la Sección 6):

1. **Definición del problema**: Entender de que tipo se trata nuestro problema, ya sea de clasificación, regresión ...
2. **Preparación de los datos**: Se trata de recopilar los datos y aplicar técnicas de transformación y preprocesamiento de datos. En este paso, los datos se suelen dividen en datos de entrenamiento y en datos de test (y en algunos casos también se coge un conjunto de datos de validación). Los primeros son para entrenar el modelo y que se ajuste a nuestros datos, y los últimos sirven para evaluar dicho modelo.
3. **Evaluación de modelos**: Se trata de evaluar los modelos de aprendizaje automático en el conjunto de datos. Requiere que se diseñe un test de pruebas robusto para poder evaluar estos modelos.
4. **Finalizar el modelo**: Seleccionar y utilización del modelo elegido en el paso anterior.
5. **Reproducir los resultados**: Dado el modelo definitivo, en este paso se reproducen los resultados. Esto implica entender e interpretar dichos resultados, para posteriormente hacer una buena conclusión.

Cuando se construyen modelos de aprendizaje automático, es esencial tener una idea precisa de su rendimiento en datos no vistos. En muchas ocasiones no se tienen muchos datos, o se quiere proporcionar una estimación imparcial y realista del desempeño del modelo. Es en este momento cuando entra en juego la **validación cruzada**.

4.2. Cross-validation

La **validación cruzada**, o *cross-validation* en inglés, es una técnica utilizada en el campo del aprendizaje automático y la estadística para evaluar el rendimiento de un modelo predictivo. Su objetivo principal es estimar cómo de bien se generalizará un modelo a datos no vistos. En este trabajo se va a hacer uso del **K-fold cross-validation**, la técnica de validación cruzada más utilizada actualmente.

El K-fold cross-validation funciona de la siguiente manera:

- Se divide el conjunto de datos en K conjuntos de tamaño aproximadamente igual.

- Se selecciona uno de los conjuntos como conjunto de prueba y los $K-1$ pliegues restantes como conjunto de entrenamiento.
- Se entrena el modelo utilizando el conjunto de entrenamiento y se evalúa su rendimiento utilizando el conjunto de prueba.
- Se repite este proceso K veces, cada vez seleccionando un conjunto diferente como conjunto de prueba.
- Se calcula la medida de rendimiento promedio utilizando los resultados obtenidos en las K iteraciones.

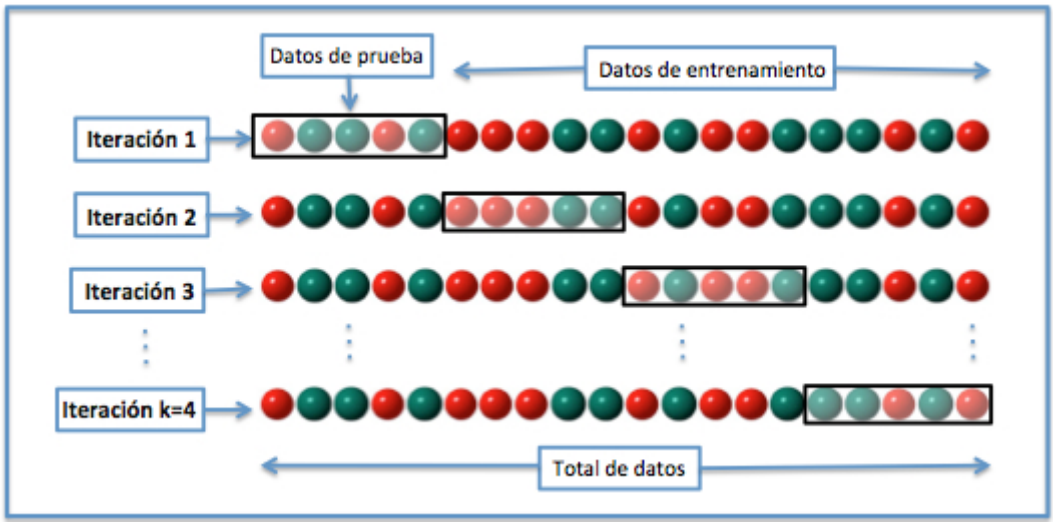


Figura 3: Validación cruzada de 4 iteraciones.

Para el resultado final, en este trabajo se van a usar la métrica **accuracy** (*exactitud*), dada por

$$Acc = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}}, \tag{1}$$

y para verificar esté método se va a comprobar que la **desviación estándar** de los accuracy de los K resultados no es muy grande, pues eso significaría que hay particiones en el dataset donde no se ajusta bien el modelo.

La principal ventaja de esta técnica es que permite aprovechar al máximo los datos disponibles, ya que se utiliza cada conjunto tanto para entrenamiento como para prueba. Esto ayuda a reducir el sesgo y la variabilidad en la evaluación del modelo. Por estas ventajas, el K-fold cross-validation es la técnica para evaluar modelos más usada cuando

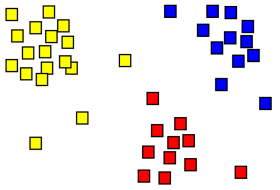
el conjunto de datos no es muy grande. Para una explicación más extensa acerca de esta técnica ir al Apéndice A.1.

Dado que este trabajo se basa en granjas no extensivas, en muchas ocasiones no hay demasiados datos. Es por esta razón que es conveniente utilizar el cross-validation a la hora de evaluar el rendimiento de un modelo.

Una vez que hemos comprendido el funcionamiento del machine learning, es crucial explorar los métodos y técnicas que conforman este campo. En este trabajo se presentan con toda clase de detalles todas las técnicas que hemos usado para construir los modelos acuerdo a los datos que tenemos. Estas técnicas son el **análisis de clústeres**, la **regresión logística**, el **SVM**, las **redes neuronales artificiales** y las **redes neuronales recurrentes**.

4.3. Análisis Clúster

El **Análisis Clúster** ("*clustering*" en inglés), también conocido como *Análisis de Conglomerados*, es una **técnica de agrupación no supervisada** que se utiliza para agrupar objetos o instancias similares en conjuntos llamados **clústeres**. De este modo, los puntos de datos que pertenecen al mismo clúster deben de tener características similares. El objetivo principal del análisis de clústeres es encontrar patrones o estructuras intrínsecas en los datos, sin la necesidad de tener información previa sobre las clases o categorías a las que pertenecen los objetos.



Existen diferentes algoritmos de clustering que se utilizan para realizar esta tarea, aunque en este trabajo únicamente se abordarán dos de ellos, el **clustering jerárquico** y el k**-medias**.

4.3.1. Clustering jerárquico

El **clustering jerárquico** es un método de análisis de datos no supervisado utilizado para agrupar objetos similares en conjuntos más grandes y comprender su estructura jerárquica. En resumen, el proceso de clustering jerárquico implica construir una estructura de agrupación en forma de árbol, donde los objetos se agrupan en subgrupos más pequeños y luego se combinan gradualmente para formar grupos más grandes.

La **distancia entre los objetos** (hiperparámetro) se calcula utilizando una medida de similitud, siendo las distancias más utilizadas las de la Figura 7 (Apéndice A.3).

En este contexto, para el cálculo de la **distancia entre los clústeres** (hiperparámetro) se utilizan los métodos de enlace, siendo los más utilizados los siguientes:

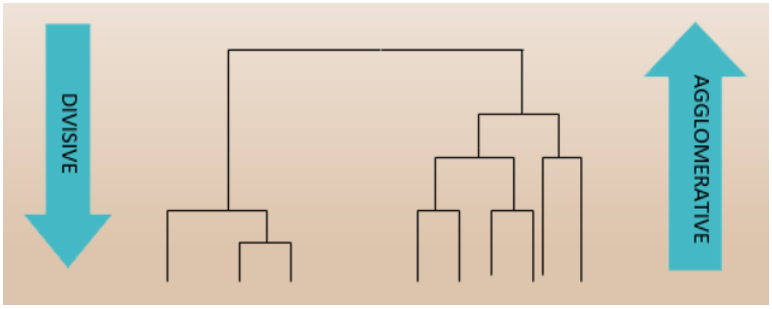
- **Average-linkage**: Dados dos grupos A y B y un par de elementos $x_i \in A$ y $x_j \in B$, este método calcula la distancia entre los dos grupos como

$$D_a(A, B) = \frac{\sum_{i,j} d(x_i, x_j)}{|A| \cdot |B|},$$

  donde $|A|$ y $|B|$ son las cardinalidades (número de elementos) de los grupos A y B, respectivamente.

- **Ward-linkage**: También conocido como *método de varianza mínima*, este método compara la suma de errores cuadrados dentro de los clústeres con la suma de los cuadrados dentro de la unión de los conglomerados, fusionándolos para minimizarlos.

Por otro lado, el clustering jerárquico se divide en dos enfoques: agrupamiento aglomerativo y divisivo (explicación en el Apéndice A.3).



En este trabajo se ha optado por la utilización del **clustering jerárquico aglomerativo**, ya que, a pesar de que este es más costoso dado que debe calcular y actualizar las distancias entre todos los clústeres en cada paso, es más fácil de utilizar debido a que no requiere especificar el número de clústeres de antemano.

Para una explicación con más detalle acerca de este método, ir al Apéndice A.3.

4.3.2. K-means

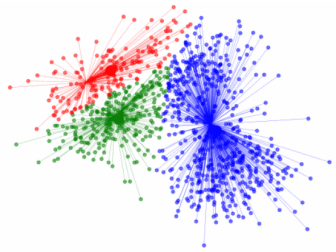
El algoritmo **k-means**, también llamado *k-medias*, es un método de agrupamiento no supervisado que busca dividir un conjunto de datos en k clústeres. El proceso se realiza de la siguiente manera:

1. Se seleccionan aleatoriamente k puntos como **centroides** iniciales.

2. Se asigna cada punto de datos al centroide más cercano basándose en la distancia previamente elegida (hiperparámetro).

3. Se recalcula la posición de cada centroide como el centroide promedio de los puntos asignados a él.

4. Se repiten los pasos 2 y 3 hasta que los centroides converjan (con cierta tolerancia) o se alcance un número máximo de iteraciones.



El objetivo de este algoritmo es minimizar la suma de las distancias cuadradas entre cada punto de datos y su centroide asignado, lo que se conoce como la **función de coste**. Al final de la ejecución del algoritmo, se obtienen k clústeres donde los puntos dentro de cada grupo son similares entre sí y diferentes de los puntos en otros grupos.

El algoritmo k-means es rápido y eficiente, pero su resultado final puede variar dependiendo de los centroides iniciales. Por lo tanto, en este trabajo ejecutaremos el algoritmo varias veces con diferentes inicializaciones y seleccionaremos la mejor solución mediante la evaluación de la coherencia de los clústeres.

Para una explicación con más detalle acerca de este método, ir al Apéndice A.4.

4.4. Regresión logística

La **regresión logística** es un método estadístico supervisado utilizado para modelar la relación entre una variable categórica y un conjunto de variables independientes. En este trabajo únicamente se utiliza para **clasificación binaria**. Es por ello que la posterior explicación de este método se centra en que únicamente hay dos categorías. Para entender perfectamente este método, así cómo entender de donde salen las fórmulas que se explican a continuación, es conveniente leer el Apéndice A.5.

El proceso de regresión logística es el siguiente:

1. Se recopilan datos de entrenamiento que contienen una variable binaria y un conjunto de variables independientes.

2. Se calcula la probabilidad de que la variable dependiente pertenezca a una de las

dos categorías utilizando la **función logística**

$$Logit(\pi_i) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p},$$

donde $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p})$ es conjunto de variables explicativas del dato i, π_i es la probabilidad de éxito y los βs son los parámetros a ser estimados.

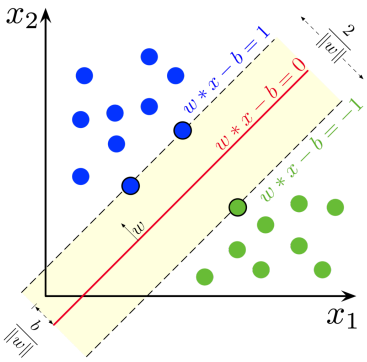
3. Se ajustan los parámetros del modelo para maximizar la probabilidad de que los datos de entrenamiento se ajusten a los valores observados de la variable dependiente.

4. Una vez ajustado el modelo, se puede utilizar para predecir la probabilidad de pertenencia a una categoría para nuevos datos.

5. Se establece un **umbral** para clasificar las predicciones como una de las dos categorías. En este trabajo se ha usado un umbral de 0.5, pues es el más obvio y común. Esto significa que si la probabilidad de que un dato pertenezca a una categoría es superior al 50 %, la predicción indicará que dicho dato pertenece a esa categoría. Por el contrario, si la probabilidad es inferior al 50 %, la predicción indicará que el dato pertenece a la otra categoría.

En cuanto al tipo de penalizaciones de este modelo, hemos elegido la **penalización** L_2 con la **constante de penalización** $C = 1$ con el fin de intentar reducir los pesos del modelo.

4.5. SVM



Las **Máquinas de Vectores de Soporte** (SVM, de sus siglas en inglés "*Support Vector Machine*"), es un **algoritmo de aprendizaje supervisado** utilizado tanto para problemas de clasificación como de regresión. Su objetivo principal es encontrar un **hiperplano** que mejor separe las diferentes clases de datos. Para una explicación extensa y detallada acerca de este algoritmo, ir al Apéndice A.6.

En nuestro caso, veremos que los datos con los que contamos no son **linealmente separables**, por tanto, no funcionará bien el hecho de encontrar un hiperplano que mejor separe las diferentes clases de datos en el espacio original. Para solucionar este caso se usa una variante de este algoritmo llamada **Kernel SVM** (ver explicación en el Apéndice A.6.1), que permite clasificar conjuntos de

datos que no son linealmente separables en su forma original. En lugar de transformar los datos explícitamente a un espacio de mayor dimensionalidad, el Kernel SVM utiliza una **función de kernel** (hiperparámetro) para calcular las similitudes entre pares de puntos de datos en un espacio de alta dimensionalidad sin la necesidad de realizar la transformación explícita.

En resumen, el Kernel SVM funciona de la siguiente manera:

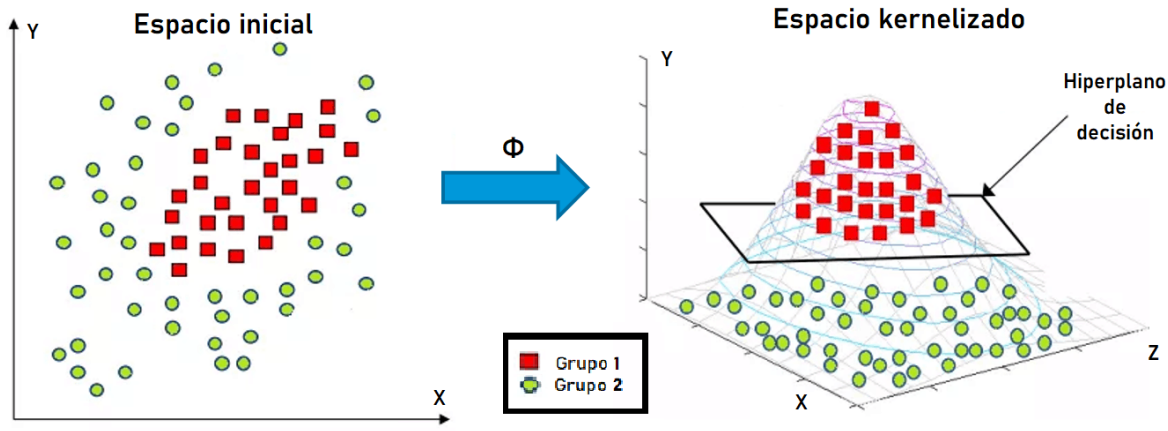
1. Recibe un conjunto de datos y una función de kernel a elegir (para ver algunas de las funciones de kernel más comunes, así como una explicación de cada una, se recomienda leer el Apéndice A.6.1).

2. Calcula la matriz de similitud que representa las distancias entre todos los pares de puntos de datos en el nuevo espacio de alta dimensionalidad (**espacio kernelizado**).

3. Entrena el modelo SVM utilizando la matriz de similitud en lugar de los datos originales.

4. Encuentra el hiperplano óptimo en el espacio kernelizado que maximice la separación entre las clases utilizando los **vectores de soporte**, es decir, los puntos de datos que se encuentran más cerca del hiperplano de separación entre las clases.

5. Clasifica nuevos datos calculando su distancia con los vectores de soporte y utilizando la información del hiperplano óptimo.



Por la naturaleza de nuestros datos, la función kernel elegida para nuestro modelo será el **Kernel RBF** (*kernel gaussiano*), debido a su capacidad para manejar relaciones no lineales en los datos. Este kernel asigna un valor de similitud o cercanía entre dos puntos

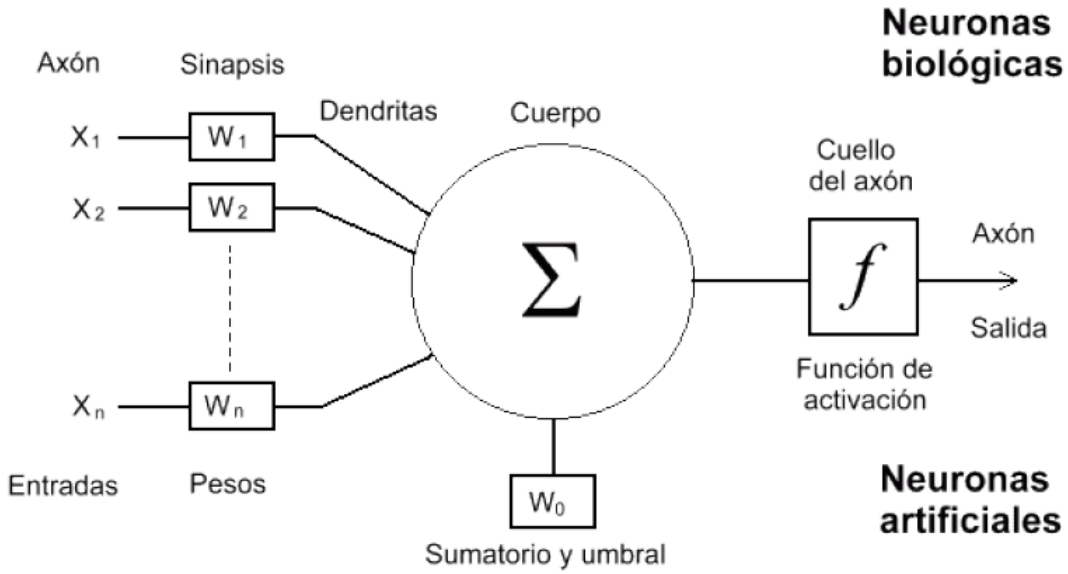
de datos. Cuanto más cercanos estén los puntos, mayor será el valor de similitud, y a medida que aumente la distancia, el valor de similitud disminuirá. Esto permite capturar relaciones complejas en los datos y lograr una clasificación más precisa.

4.6. Redes neuronales artificiales

Las redes neuronales son modelos de aprendizaje automático utilizados en el campo de la **neurocomputación**. Este campo busca desarrollar algoritmos inspirados en los procesos cerebrales. Su principal objetivo es el reconocimiento de patrones, y han supuesto un avance significativo en áreas donde los algoritmos de aprendizaje tradicionales solían fallar con frecuencia, como el procesamiento del lenguaje natural o el aprendizaje visual de patrones.

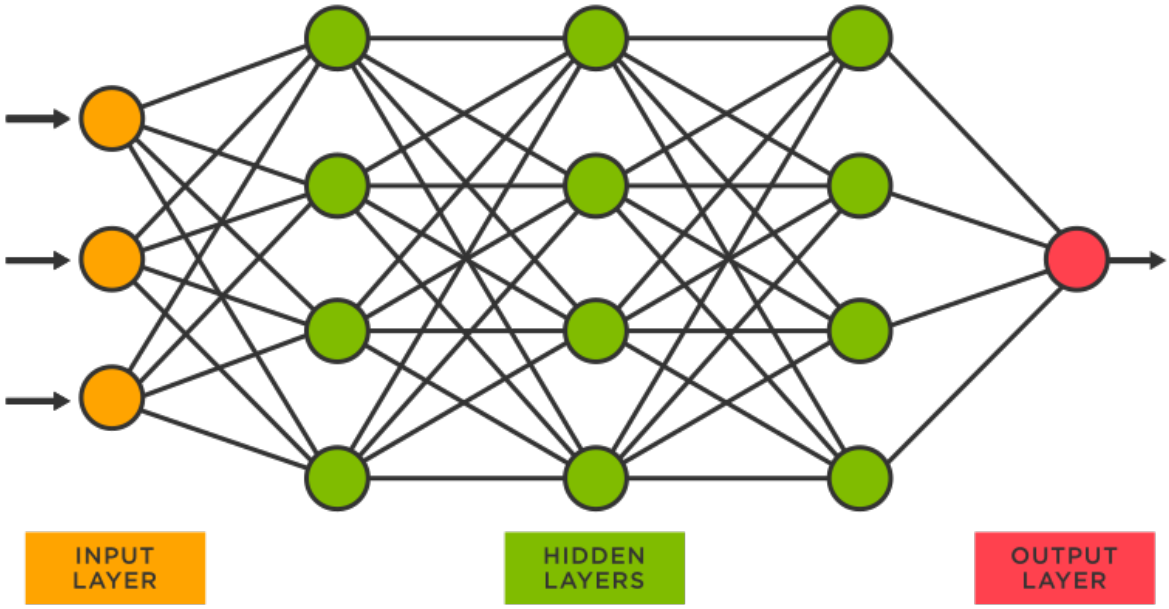
Existen distintos modelos de redes neuronales, cada uno especializado para un tipo de problemas. En este trabajo se presentan los siguientes modelos: las **redes neuronales artificiales** y las **redes neuronales recurrentes**. A continuación se proporcionará una breve explicación de cada uno de estos modelos. Si se desea profundizar más acerca de estos modelos de redes neuronales, se recomienda ir a los apéndices A.7 y A.8.

Una **red neuronal artificial** o ANN (por sus siglas en inglés *Artificial Neural Network*) es un modelo matemático y computacional que se inspira en el funcionamiento de las redes neuronales biológicas presentes en el cerebro humano.



Las ANN funcionan mediante la interconexión de neuronas artificiales en **capas**, divididas en tres tipos principales:

- **Capa de entrada** (*input layer*): Es la capa inicial de la red neuronal que recibe los datos de entrada y los transmite a las capas ocultas. Cada nodo representa una característica de los datos de entrada.
- **Capas ocultas** (*hidden layers*): Son capas intermedias entre la capa de entrada y la capa de salida. Realizan operaciones matemáticas y aplican funciones de activación para procesar la información y extraer características relevantes de los datos. La elección de estas capas, así como las neuronas en cada capa, es de lo más complicado e importante para encontrar la red que mejor se ajuste a los datos que se tienen.
- **Capa de salida** (*output layer*): Es la capa final de la red neuronal y produce los resultados finales. Puede tener una sola neurona (para clasificación binaria) o varias neuronas (para clasificación multiclase o regresión), y representa las salidas deseadas de la red.



Cada neurona recibe entradas de otras neuronas, las procesa y produce una salida. Las conexiones entre las neuronas tienen **pesos** asociados que determinan la importancia de cada entrada. El proceso de una red neuronal se puede resumir en los siguientes pasos:

1. **Propagación hacia adelante** (*Feedforward*): Las entradas se propagan a través de la red neuronal desde la capa de entrada hasta la capa de salida. Cada neurona suma las entradas ponderadas por sus pesos, las procesa mediante una **función de activación** y produce una salida.

2. **Cálculo del error**: Se compara la salida producida por la red con la salida deseada para determinar la diferencia, conocida como error. El objetivo es minimizar este error para que la red pueda producir salidas más precisas.

3. **Retropropagación del error** (*Backpropagation*): El error se propaga hacia atrás a través de la red neuronal, ajustando los pesos de las conexiones para reducir el error en cada neurona. Este proceso utiliza algoritmos de optimización, como el descenso del gradiente, para actualizar los pesos de manera que la red aprenda de los datos de entrenamiento.

4. **Actualización de pesos**: Los pesos de las conexiones se actualizan iterativamente utilizando el **algoritmo de optimización** elegido. El objetivo es encontrar los pesos que minimicen el error en las salidas de la red neuronal.

5. **Repetición del proceso**: Los pasos 1 a 4 se repiten varias veces, conocidas como **épocas** o número de entrenamientos, hasta que la red neuronal converge (con cierta tolerancia) o llega a un número máximo de iteraciones y produce salidas precisas para los datos de entrada.

Para una explicación más extensa acerca del algoritmo de entrenamiento de las ANN se recomienda leer el Apéndice A.7.3.

Tal y como se observa en el proceso de entrenamiento, existen muchos hiperparámetros a elegir a la hora de crear una ANN. A continuación se proporciona una breve explicación de cada hiperparámetro, así cómo los usados en este trabajo.

- **Funciones de activación**: son funciones utilizadas por cada neurona para determinar la salida en función de las entradas recibidas. Estas funciones pueden introducir no linealidades en el modelo, lo que permite que la red neuronal pueda aprender relaciones y patrones más complejos en los datos. Algunas de las funciones de activación que se han usado en este trabajo incluyen la función **ReLU** (Rectified Linear Unit) y la función **sigmoide** (ver en la Tabla 4). Para una explicación detallada de cómo y cuando utilizar estas y otras funciones ir al Apéndice A.7.2

- **Algoritmos de optimización**: estos algoritmos se utilizan para ajustar los pesos de las conexiones entre las neuronas con el fin de minimizar el error de la red y mejorar su rendimiento. Uno de los algoritmos de optimización más utilizado es el **método del descenso del gradiente** (*SGD*) (ver [1]), que busca encontrar el mínimo global de la función de pérdida mediante la iteración y ajuste de los pesos

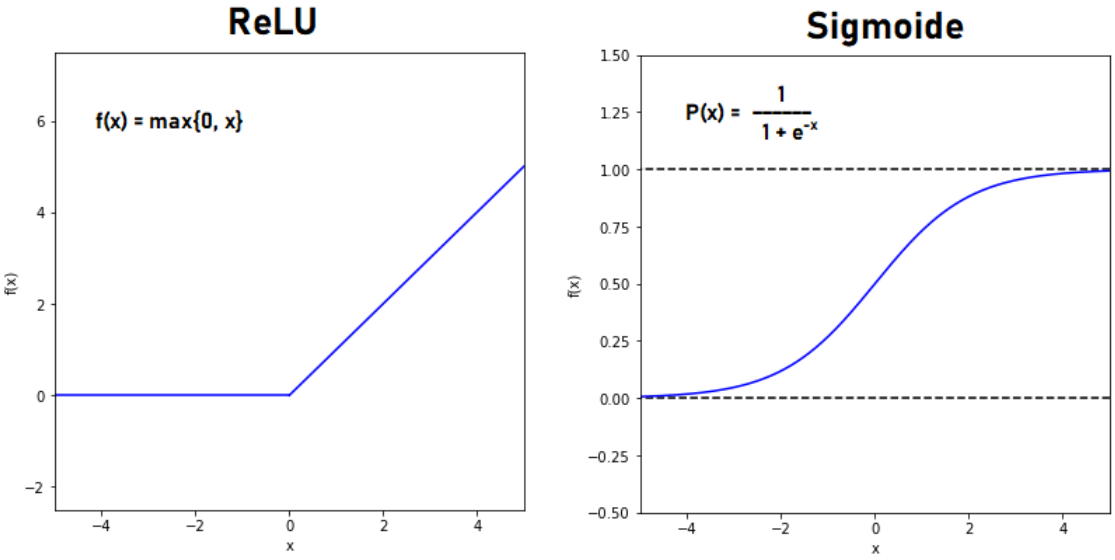
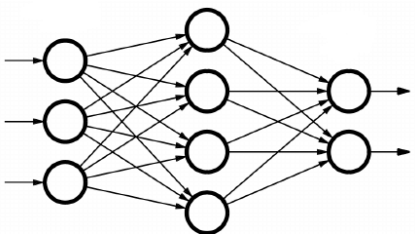


Figura 4: Algunas de las funciones de activación más comunes.

en dirección opuesta al gradiente. Otro algoritmo que utilizaremos en este trabajo debido a su rapidez es la **estimación del momento adaptativo** (*ADAM*, por sus siglas en inglés **adaptive moment Estimation**) (ver [16]).

- **Épocas**: elegir el número de entrenamientos que queremos que realice nuestra red es uno de los parámetros más complicados de elegir, ya que depende de muchos factores como el número de datos que tengamos o la rapidez de convergencia del método, entre otros. Por otro lado, es uno de los hiperparámetros más importantes, pues si elegimos pocas épocas se produce el efecto llamado **underfitting**, y si se eligen demasiadas épocas se produce el efecto contrario, el conocido **overfitting**. Para una explicación de estos efectos ir al Apéndice A.2.

4.6.1. Redes neuronales feedforward



En este trabajo se usarán un tipo específicas de ANN, las **redes neuronales feedforward**, también conocidas como redes neuronales de propagación hacia adelante. Estas son un tipo de arquitectura de redes neuronales artificiales que siguen una dirección unidireccional del flujo de información, de manera que la información se propaga desde la capa de entrada a través de una o varias capas ocultas hasta llegar a la

capa de salida, sin conexiones retroactivas.

Estas redes funcionan como aproximadores universales de funciones (ver [6]) y son capaces de generar representaciones intermedias de los datos que otros algoritmos no podrían obtener sin conocimiento humano previo.

En este contexto, las redes neuronales artificiales son muy útiles y versátiles, y se utilizan en una amplia variedad de aplicaciones en diferentes campos. Pero para el análisis secuencial, como el **análisis de series temporales**, estas redes no suelen funcionar muy bien. Aquí es donde entran en juego las **redes neuronales recurrentes**.

4.7. Redes neuronales recurrentes

Las **Redes Neuronales Recurrentes** (RNN, de sus siglas en inglés "*Recurrent Neural Networks*") son un tipo especializado de modelo de aprendizaje automático diseñado para procesar datos secuenciales. A diferencia de las redes neuronales convencionales, las RNN tienen la capacidad de mantener y utilizar información de eventos previos en la secuencia para influir en la salida actual. Esta capacidad de "memoria" hace que las RNN sean especialmente adecuadas para tareas que involucran datos secuenciales, como el procesamiento del lenguaje natural, el reconocimiento de voz, la traducción automática y la generación de texto, entre otros.

Las RNN son capaces de capturar relaciones temporales complejas y modelar dependencias a largo plazo en los datos secuenciales. Esto se logra mediante la retroalimentación recurrente, donde la salida anterior de la red se utiliza como entrada en el siguiente paso de tiempo, lo que permite que la información fluya a través de la secuencia y se utilice en la toma de decisiones posteriores. Esta propiedad las diferencia de las redes neuronales convencionales, que tratan cada entrada de manera independiente y no tienen en cuenta el contexto temporal.

A pesar de sus características atractivas, las RNN no se han convertido en una herramienta convencional en el aprendizaje automático debido a la dificultad de entrenarlas de manera efectiva. Esto se debe a la relación inestable entre los parámetros y la dinámica de los estados ocultos, que se manifiesta en el **problema del decaimiento del gradiente** (ver en [10]). Como resultado, ha habido poca investigación sobre el modelo RNN estándar en los últimos 20 años, y solo se han visto algunas aplicaciones exitosas que utilizan RNN grandes ([11], [12]).

Para evitar el problema del decaimiento del gradiente en las RNNs tradicionales, fueron diseñadas un tipo especial de RNN llamadas **LSTM** (*Long Short-Term Memory*), que

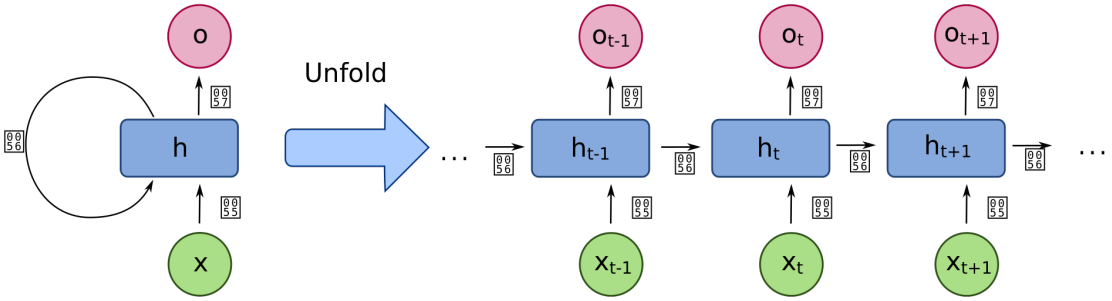


Figura 5: Despliegue de una red neuronal recurrente.

son las que utilizaremos en este trabajo. Para una explicación de la arquitectura y del funcionamiento de las LSTM ir al Apéndice A.8.

Además de las RNN tradicionales y las LSTM, existen otros tipos de estas redes, como pueden ser las **GRU** (*Gated Recurrent Unit*) o las **RNN bidireccionales**. Dado que estas redes no se utilizan en este trabajo tampoco se van a explicar, pero si el lector quiere información acerca de estas, se recomiendan las lecturas [3] para las GRU y [13] para las RNN bidireccionales.

Ahora que ya tenemos una idea acerca de para qué sirven y cómo funcionan estos métodos, veamos cuáles son los datos con los que partimos en este trabajo, así cómo la transformación y el preprocesamiento de estos.

5. Datos

En una granja, se pueden extraer diversos tipos de datos relacionados con diferentes aspectos de la operación y el manejo de los animales. Algunos de los tipos de datos comunes que se pueden recopilar en una granja incluyen datos de posición, de alimentación, de actividades o ambientales.

En nuestro caso, los datos de partida para la realización del proyecto vienen dados por un archivo en formato "csv" que recoge la información detallada sobre las actividades llevadas a cabo por cada ternero a lo largo del tiempo, así como también indica el grupo alimenticio al que pertenece cada uno de ellos.

De esta forma, nuestra tabla de datos contiene 1045945 filas y 9 campos o columnas, las cuales son:

- **Hora de inicio local**: Hora exacta en la que se registran los datos.
- **ID de ganado**: Identificación del ganado.
- **Número del animal**: Número que se le ha asignado a cada animal.
- **Nombre de la sucursal**: Explicación de si es un ternero lactante o no.
- **Número del grupo**: Número al que se le ha asignado cada grupo alimenticio (ver en la Tabla 1).
- **Nombre del grupo**: Nombre del grupo alimenticio.
- **Número del estado**: Número al que se le ha asignado cada actividad/estado (ver en la Tabla 2).
- **Nombre del estado**: Nombre de la actividad/estado.
- **Duración**: Intervalo de tiempo en el que se han registrado los datos, en minutos.

Número	Grupo	Descripción
1	Terneras 2 tomas	Se les aporta leche 2 veces al día
2	Destetadas	Terminan con la lactancia / Dejan de tomar leche
3	Terneras 3 tomas	Se les aporta leche 3 veces al día

Tabla 1: Descripción de los grupos

Número	Estado	Descripción
1	Baja actividad	Estado de baja actividad
2	Media actividad	Estado de media actividad
3	Alta actividad	Estado de alta actividad
4	Rumiando	Regurgitación del material ingerido
5	Comiendo	Estado de comer
6	Andando	Estado de andar
7	Pastando	Comer pasto
8	Over heat	Estado de sobrecalentamiento
9	No rumiando	Estado donde no regurgita el material ingerido
10	Lactando	Amamantar o criar con leche a una cría
12	Oral no nutritiva	Comer material no nutritivo
15	Inválido	Actividades que no se tiene interés en predecir

Tabla 2: Descripción de los estados

El tratamiento de nuestros datos pasa primero por una fase de limpieza, donde se van a eliminar las celdas cuyo estado es *Inválido*, pues son aquellas a las que no se tiene interés en predecir (NA).

Dado que este dataset provienen de granjas no extensivas, únicamente contiene información de 58 terneras, lo cual es una de las complicaciones con la que tenemos que lidiar a la hora de trabajar con modelos de machine learning, pues es bastante complicado realizar estos modelos con tan pocos datos.

A pesar de esto, de muchas de estas terneras se tiene información de largos períodos de tiempo, de modo que muchas pasan por más de un grupo alimenticio. De este modo, se tienen 33 terneras de 2 tomas, 27 de 3 tomas y 53 destetadas, qué sumando estos tres conjuntos se obtiene una cantidad de 113 animales de su respectivo grupo.

Para los objetivos del trabajo vamos a necesitar crear dos nuevos conjuntos de datos a partir de los datos que tenemos, uno para **clasificación** y otro para **análisis de series temporales**.

5.1. Dataset para la clasificación

Dado que 113 animales de su respectivo grupo siguen siendo muy pocos, se han separado todos estos animales por días. Esto significa que si por ejemplo tenemos una ternera

destetada con datos de un mes, entonces lo dividiremos de forma que tendremos 30 nuevos "animales" con datos de sus actividades a lo largo de un día. Este procedimiento se ha pensado para ampliar los datos con los que trabajar. Con este procedimiento tenemos 1160 entradas correspondientes a terneras de 2 tomas, 861 de 3 tomas y 1169 destetadas, qué sumando estos tres conjuntos se obtiene una cantidad de 3190 entradas con los que vamos a trabajar, lo cual ya es una cantidad bastante razonable de datos para poder empezar a realizar modelos. Cabe destacar que se ha puesto una tolerancia de 12 horas para generar los datos por días, lo que significa que si los datos que se tienen de un día no superan estas 12 horas, entonces no se añade en nuestro nuevo dataset. Esto se ha hecho para no contar a los animales que se les han registrado muy pocos datos, pues no serían entradas muy fiables.

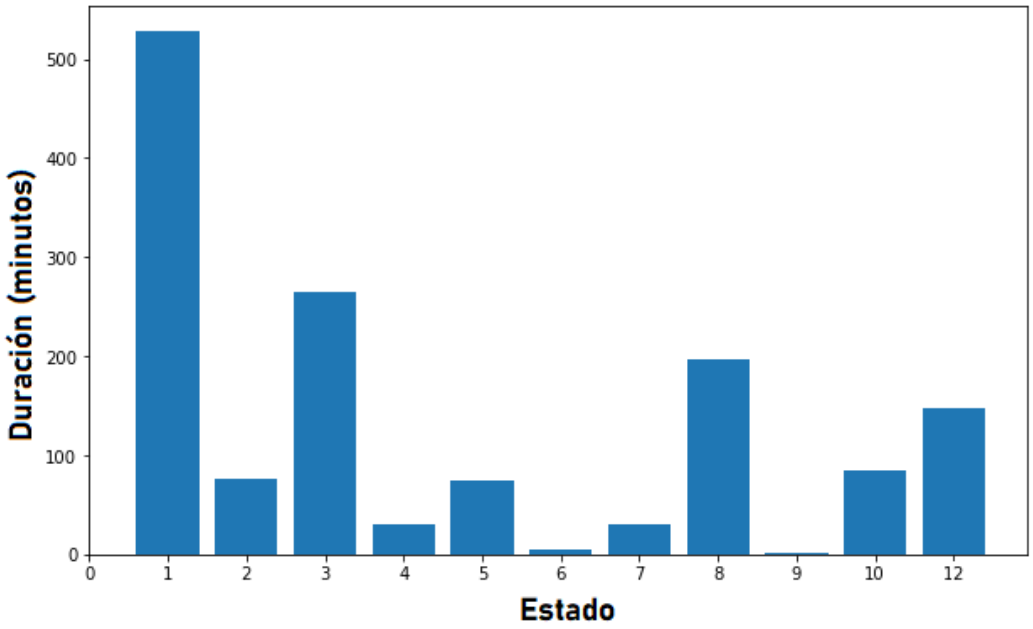


Figura 6: Diagrama de barras de las actividades diarias de una ternera de 2 tomas.

Para nuestro primer objetivo, que recordamos que es clasificar los animales por grupos alimenticios a través de sus actividades diarias, el nuevo dataset que se ha creado a partir de estos datos tiene los siguientes campos/columnas:

- **Número del animal**: Número que se le ha asignado a cada animal en el dataset inicial.
- **Grupo del animal**: Número del grupo alimenticio del animal.
- **Día**: Día en el que se han registrado las actividades del animal.

- **Vector de estados**: Vector de las duraciones (en minutos) de todas las actividades que ha hecho el animal durante un día. Tal y como se ha explicado anteriormente, la suma de estas duraciones es mayor a 12 horas. Este vector es el que se ha graficado en la Figura 6.
- **Vector de estados normalizados**: Vector de estados donde la suma total es igual a 1. Eso te ha hecho porque en muchos casos no se tenían los datos de un día entero, ya sea por falta de datos o por quitar el estado *Inválido*.

En este caso, para identificar a un animal no basta con su número, sino también es necesario conocer su grupo y el día.

5.2. Dataset para el análisis de series temporales

Para nuestro segundo objetivo, que recordamos que es predecir las actividades que realizará una ternera perteneciente a un grupo específico, el nuevo dataset no se separará por días, pues cuanto más largo sea el intervalo de tiempo en el que se han registrado las actividades mejor a la hora de hacer predicciones temporales. Las unidades de tiempo más pequeñas que han usado en el dataset inicial han sido los minutos, por lo que vamos a usar los minutos como unidad de tiempo en el nuevo conjunto de datos. Por tanto, este dataset se ha construido a partir de los siguientes campos/columnas:

- **Número del animal**: Número que se le ha asignado a cada animal en el dataset inicial.
- **Grupo del animal**: Número del grupo alimenticio del animal.
- **Tiempo**: Tiempo en el que se ha registrado la actividad del animal, en minutos.
- **Estado**: Estado del animal en el minuto especificado por los campos anteriores.

Esta tabla no está exenta de errores, pues las observaciones no están registradas a intervalos regulares de tiempo, es decir, existen varios intervalos temporales donde no se han registrado los datos o se han registrado como *Inválido*, tal cómo se puede observar en la Figura 7. Este problema dificulta la aplicación de técnicas de análisis secuenciales diseñadas para datos con intervalos regulares.

Una de las mayores complicaciones al realizar este tipo de análisis es que la mayoría de técnicas están diseñadas para datos continuos (como ARIMA ([4]) o modelos de suavizado

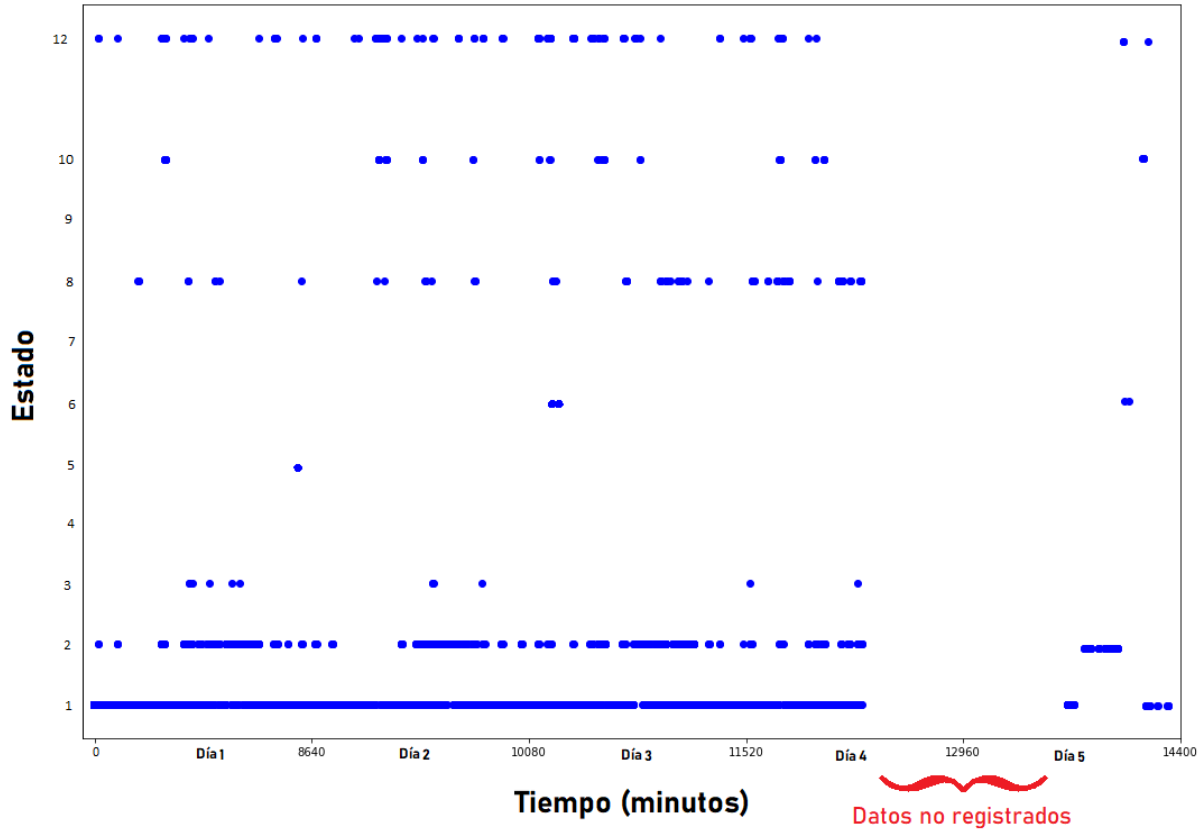


Figura 7: Gráfica de una ternera destetada usando el dataset secuencial.

exponencial), pues es más fácil interpolar y extrapolar datos representados por una función continua que datos discretos, y en este caso los datos del estado son discretos. Por tanto, identificar patrones estacionales y ciclos en estos datos puede ser bastante desafiante.

Otra complicación es que, dado que los estados se califican del 1 al 12 (quitando el 11), la predicción de estos valores es una tarea muy complicada y con un margen de error grandísimo, con una probabilidad de acierto de tan solo $100 \cdot \left(\frac{1}{11}\right) = 9.09\,\%$.

Creados estos datasets, ahora ya podemos empezar a realizar los modelos que nos ayuden a conseguir nuestros objetivos.

6. Desarrollo y resultados

Todo el desarrollo de este trabajo que se explica a continuación se ha realizado mediante **Python**, un entorno de programación que se caracteriza por su facilidad de uso, así como por su popularidad y, como consecuencia de esta, por su gran comunidad de desarrolladores. Python es utilizado a nivel mundial como software para modelos de IA, por lo que cuenta con una gran cantidad de librerías especializadas en el ML, como son el caso de *TensorFlow*, *Keras* y *scikit-learn*, utilizadas en este trabajo. En el Apéndice B se puede encontrar una expliación acerca de las librerias utilizadas en el trabajo.

6.1. Modelo de clasificación

Para clasificar los 3190 animales en sus correspondientes grupos alimenticios vamos a empezar con un **análisis de clústeres** (no supervisado), para ver si podemos identificar relaciones intrínsecas en las actividades diarias de los animales sin la necesidad de conocer previamente los grupos a los que pertenece cada uno.

Usando el **clustering jerárquico aglomerativo**, el método de enlace que más buen resultado nos ha dado es el *ward-linkage*, el cual requiere que la métrica de distancia sea la *euclidiana*, que observando la Tabla 7 es la que más sentido tiene aplicar a nuestros datos.

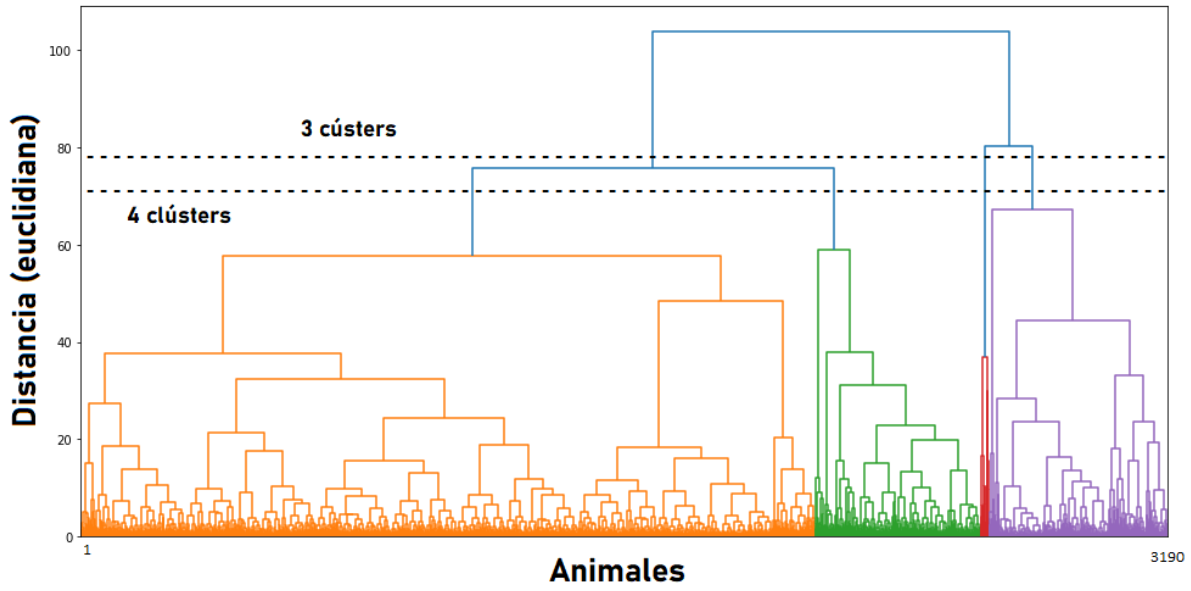


Figura 8: Dendograma con el método *ward-linkage*.

Tal y como se observa en la Figura 8, si elegimos 3 clústeres, que son el número de grupos alimenticios que tenemos, nos queda un clúster con tan solo 24 animales. Si bajamos muy poco la distancia entre clústeres enseguida aparece otro, por tanto, hemos elegido un número final de 4 clústeres. Comparando estos clústeres con los animales etiquetados obtenemos la Tabla 3.

Clúster	**Terneras 2 tomas**	**Terneras destetadas**	**Terneras 3 tomas**
1	1104	281	772
2	46	357	82
3	0	24	0
4	10	507	7

Tabla 3: Comparación del clúster jerárquico con los animales etiquetados.

Como resultado no obtenemos una muy buena clasificación. Pues el grupo de destetadas sí que es capaz de distinguirlo más o menos de los otros en los clústeres 3 y 4, en cambio, los grupos de 2 tomas y 3 tomas no los consigue clasificar nada bien, lo cual nos indica que tienen una estructura jerárquica similar.

A continuación probamos con un método de clustering basado en centroide, como es el K**-means**. Dado que tenemos 3 grupos alimenticios, vamos a usar este método con $K = 3$ clústeres.

Tal y como se explica en la sección 4.3.2, dado que el resultado final puede variar dependiendo de los centroides iniciales, se sugiere ejecutar el método varias veces con diferentes inicializaciones y seleccionar la mejor solución mediante la evaluación de la coherencia de los clústeres. Para este procedimiento se ha usado un método de inicialización llamado "*greedy k-means++*" ([2]), el cual realiza varios ensayos en cada paso de muestreo y elige el mejor centroide entre ellos. Este método selecciona los centroides de clúster iniciales utilizando un muestreo basado en una distribución de probabilidad empírica de la contribución de los puntos a la inercia general, lo que implica una convergencia más rápida. Como resultado se ha obtenido la Tabla 4.

De un modo parecido al clúster jerárquico, vemos que el único grupo que solamente existe un clúster, el 2, que es capaz de clasificar correctamente el grupo de destetadas, en cambio, los otros clústeres no son capaces de clasificar nada bien los grupos. Para ver que las características que diferencian el clúster 3 de los otros, en la figura 9 se puede observar el diagrama de barras de los animales centroides de los clústeres.

De este modo, podemos decir que los animales que se encuentran dentro del clúster 3

Clúster	Terneras 2 tomas	Terneras destetadas	Terneras 3 tomas
1	416	483	319
2	735	320	536
3	9	366	6

Tabla 4: Comparación del método 3-kmeans con los animales etiquetados.

se diferencian de los otros en que son más activos, ya que pasan menos tiempo en baja actividad (estado 1) que los animales presentes en los otros clústeres. Y dado que dentro de este clúster un $\frac{366}{381} = 96\,\%$ son terneras destetadas, podemos afirmar, sin pérdida de generalidad, que las terneras destetadas son más activas que las de 2 o 3 tomas.

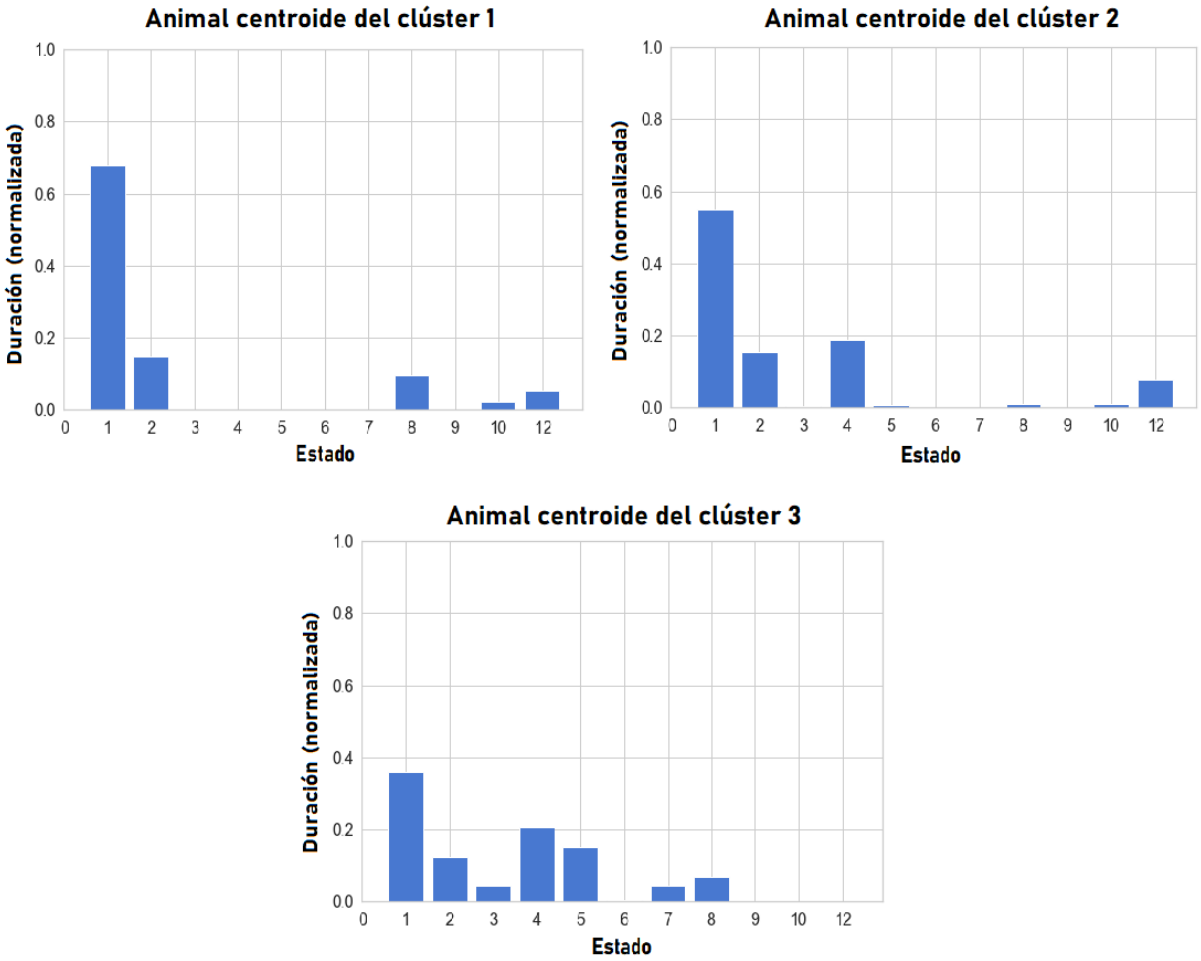


Figura 9: Diagrama de barras de los *centroides* de los clústeres del método 3-means.

Como conclusión de estos métodos de clustering podemos decir que el grupo de des-

tetadas sí que parece tener algunas características que se diferencien de los otros grupos, en cambio, los grupos de 2 y 3 tomas no parece que tengan muchas características diferenciables en las actividades del animal.

Dado que con aprendizaje no supervisado no se han podido clasificar correctamente los grupos, vamos a probar con aprendizaje supervisado, aprovechando que conocemos los grupos (etiquetas) de cada animal.

Nota: Para todos los resultados que se van a sacar a partir de ahora se va a usar el método de **cross-validation con** 5 **conjuntos/iteraciones**.

El primer método de clasificación (agrupamiento supervisado) que vamos a usar es el **Support Vector Machine**. Para ello, veamos que aspecto tienen estos datos para ver qué técnica dentro del SVM es preferible usar. Si dibujamos todos los animales según sus estados de actividad (baja, media o alta) (Figura 10), podemos observar que encontrar un hiperplano que separe estos grupos de forma lineal es prácticamente imposible.

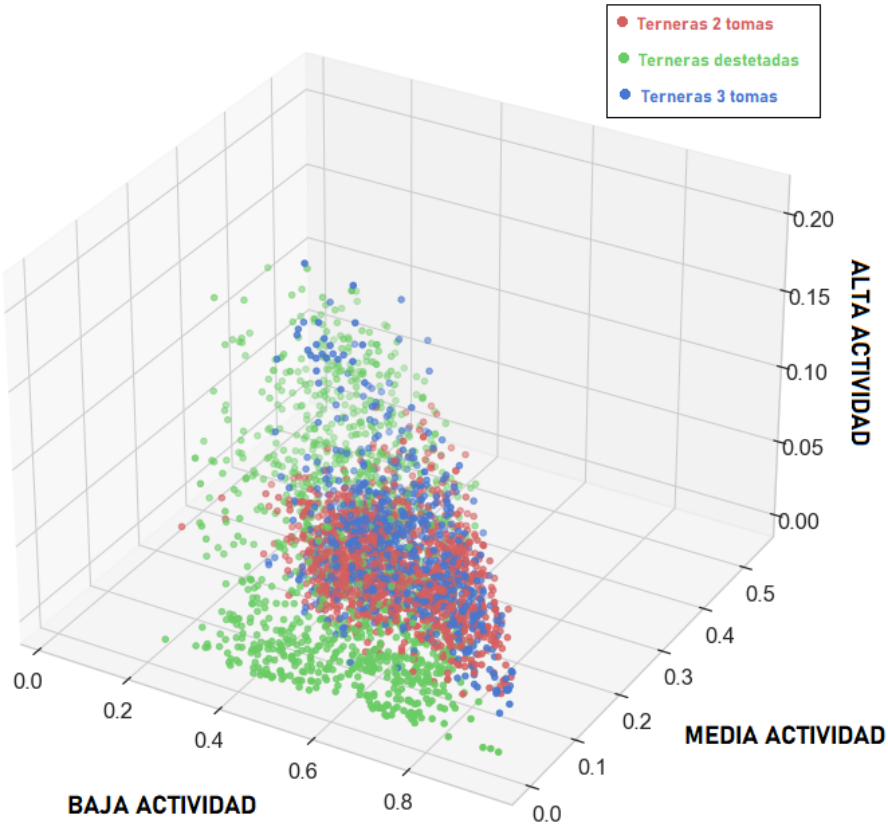


Figura 10: Gráfica 3d de los estados de actividad de los terneros con su respectivo grupo.

Nota: En esta gráfica 3d se puede observar lo mencionado anteriormente, y es que las terneras más activas son casi todas destetadas, en cambio, las terneras de 2 y 3 tomas no se diferencian prácticamente en cuanto al estado de la actividad. Cabe destacar también que hay un conjunto bastante grande de terneras destetadas que no están con alta actividad en casi ningún momento. En este caso, muchas de estas terneras están en estado de sobrecalentamiento (*over-heat*) en un largo período del día, pero también hay otras que no han pasado por ningún estado de sobrecalentamiento a lo largo del día. Esto puede ser por varias razones, entre ellas que se hayan puesto enfermas al terminar con la lactancia. Por tanto, el hecho de que una ternera destetada esté inactiva y no haya pasado por ningún estado de sobrecalentamiento, podría ser indicio de que esté pasando por una enfermedad.

Para comprobar que, efectivamente, los datos no son linealmente separables, se ha aplicado el *soft-margin SVM lineal* con parámetro de regularización $C = 1$, obtenemos la matriz de confusión

		PREDICCIONES 1	2	3
GRUPOS	1	808	185	167
	2	452	636	81
	3	543	131	187

con un *accuracy* de 51.13 % y una *desviación estándar* (de los resultados del 5-fold cross-validation) de 0.061, lo cual es un resultado bastante malo, tal y como se esperaba.

Para una mejora de estos resultados vamos a usar el **kernel SVM**. Se ha probado con varios kernels, entre ellos el polinomial y el RBF, dando un mejor resultado el polinomial de grado 2 con constante $c = 0$, con parámetro de regularización $C = 1.35$ y con penalización L_2. La matriz de confusión resultante es

		PREDICCIONES 1	2	3
GRUPOS	1	1015	51	94
	2	222	901	46
	3	657	93	111

con un *accuracy* de 63.54 % y una *desviación estándar* de 0.03, lo cual mejora respecto al resultado anterior, pero sigue siendo un accuracy bastante bajo.

Al analizar la matriz de confusión, vemos que este método logra una clasificación bastante precisa del grupo de destetados, con un acierto de predicción de $\frac{901}{1045} = 86\,\%$. Sin embargo, sigue teniendo dificultades para clasificar adecuadamente los grupos de 2 y 3 tomas, al igual que sucedía en el agrupamiento no supervisado.

Estos resultados sugieren que es probable que los grupos lactantes tengan un comportamiento similar, ya sea que estén con 2 o con 3 tomas diarias. Por esta razón, vamos a tratar los grupos de 2 y 3 tomas como un mismo grupo, llamado **grupo de terneras lactantes**, y ver si de esta forma se puede realizar una buena clasificación entre este grupo y el grupo de destetadas. Con esta nueva agrupación tenemos un total de 2021 terneras lactantes (número 1) y 1169 terneras destetadas (número 2).

6.1.1. Modelo de clasificación binaria

Al tener únicamente 2 grupos en el nuevo dataset, esto nos permite utilizar la **regresión logística binaria** como método de clasificación.

En primer lugar, vamos a encontrar los βs de la función logística

$$Logit(\pi) = \beta_0 + \beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_{12} S_{12},$$

siendo $S = (S_1, S_2, \ldots, S_{12})$ es vector de estados normalizados realizados por el animal y π la probabilidad de éxito.

A este método vamos a añadirle una penalización L_2 con constante $C = 1$, con el fin de intentar reducir los pesos del modelo. Con estos parámetros obtenemos la siguiente función logística

$$\begin{aligned} Logit(\pi) = \; & 0.53 - 0.61 S_1 + 0.34 S_2 - 0.22 S_3 + 0.41 S_4 + 0.67 S_5 + \\ & + 1.9 S_6 + 5.56 S_7 - 0.46 S_8 + 1.6 S_9 + 0.05 S_{10} - 0.27 S_{12}, \end{aligned} \tag{2}$$

cuya matriz de confusión viene dada por

		PREDICCIONES 1	PREDICCIONES 2
GRUPOS	1	1831	190
	2	680	489

con un *accuracy* de 72.73 % y una *desviación estándar* de 0.043.

Para mejorar este resultado vamos a modificar la regresión logística de modo que la función no sea lineal, para capturar las relaciones no lineales de los datos.

Primero que todo, por la ecuación 2 tenemos que el coeficiente del estado 10 (estado de lactancia) es muy pequeño, de tan solo 0.05, y por la matriz de correlaciones dada por la Figura 11 tenemos también que el estado 10 está muy poco correlacionado con el grupo del animal. Esto significa que este estado no nos influye casi a la hora de clasificar estos animales, por tanto, para ajustar más los otros parámetros se ha decidido quitarlo de la función.

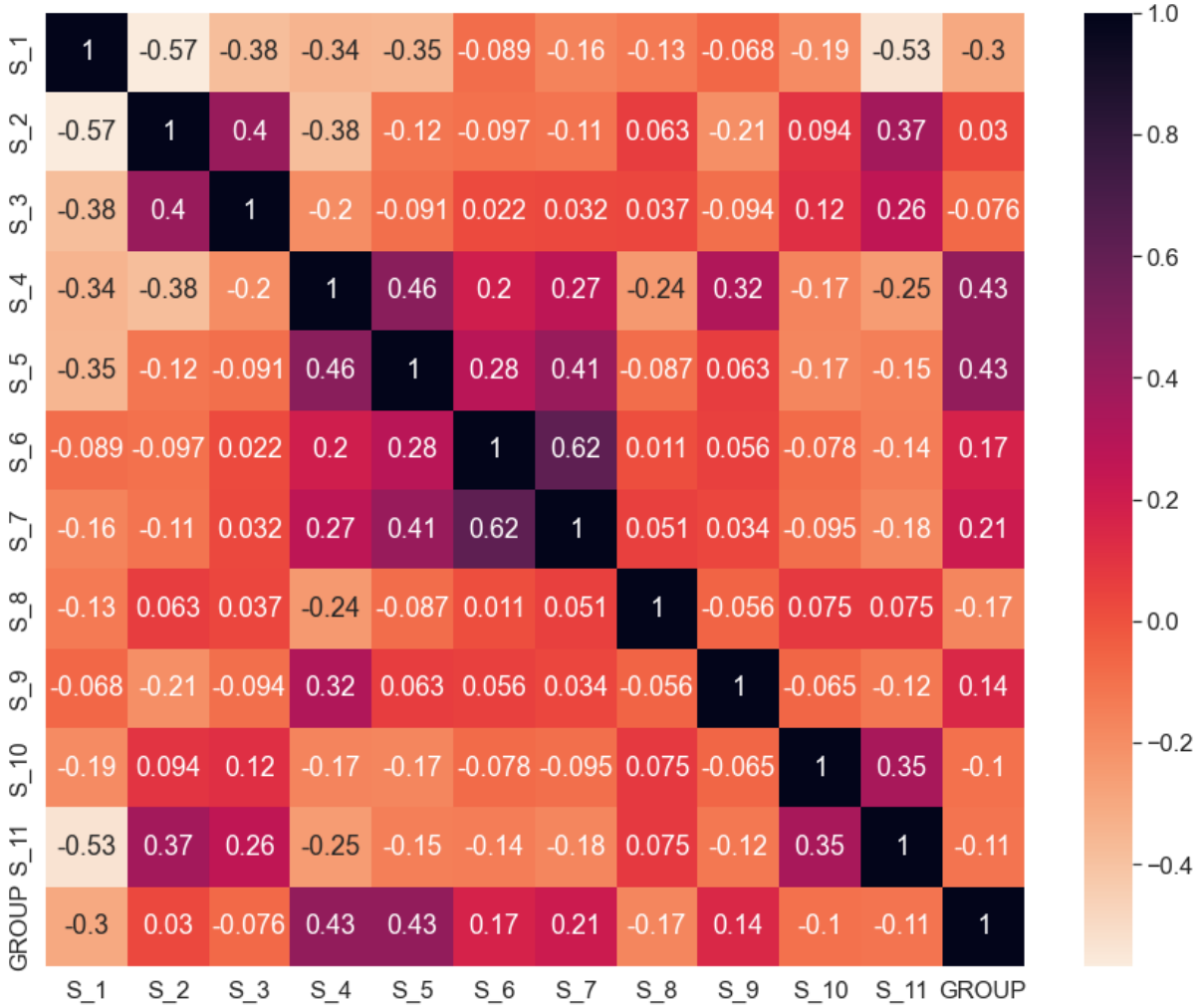


Figura 11: Matriz de correlaciones de los estados.

Por otro lado, usando la matriz de correlaciones para ver qué estados están más relacionados y haciendo pruebas, se ha concluido que la función logística que más se adapta

a nuestros datos es la siguiente

$$\begin{aligned} Logit(\pi) = \; & 0.13 - 0.19S_1 + 1.77S_2 - 1.13S_3 + 0.95S_4 + 0.78S_5 + \\ & + 1.46S_6 + 3.81S_7 - 0.76S_8 + 0.67S_9 - 0.42S_{12} - 1.4S_1S_2 - \\ & - 2.09S_1S_3 - 0.92S_1S_{12} - 1.07S_4S_5 - 1.24S_1S_2S_4, \end{aligned} \tag{3}$$

cuya matriz de confusión viene dada por

		PREDICCIONES 1	PREDICCIONES 2
GRUPOS	1	1859	162
	2	636	533

con un *accuracy* de 74.99 % y una *desviación estándar* de 0.051.

En este caso se puede observar una ligera mejora respecto al método anterior, pero aun así no se consigue una buena clasificación de ambos grupos. Una de las razones de esto es que en este conjunto de datos hay considerablemente más animales (aproximadamente el doble) en el grupo de lactantes en comparación con el grupo de destetadas.

En este momento, usando métodos de machine learning tradicional hemos conseguido una clasificación en grupos lactantes y destetados con un 74.99 % de accuracy. Para intentar mejorar este resultado vamos a probar con redes neuronales, en particular, con **redes neuronales artificiales**.

6.1.2. Redes neuornales artificiales para clasificación binaria

Para la creación de nuestra ANN vamos a usar **capas densas**, es decir, conjuntos de neuronas donde cada neurona en una capa está conectada a todas las neuronas en la capa anterior y a todas las neuronas en la capa siguiente. Hemos elegido este tipo de capas, ya que son fundamentales en muchas arquitecturas de redes neuronales, pues son esenciales para aprender representaciones complejas a partir de los datos de entrada

Por otro lado, tenemos 11 posibles estados, por tanto, tendremos 11 neuronas en la capa de entrada, y puesto que se trata de un problema de clasificación binaria tendremos únicamente una neurona en la capa de salida. Esto se debe a que la salida de esta neurona puede interpretarse como la probabilidad de pertenecer a uno de los dos grupos, en nuestro caso al grupo de destetadas. De este modo, nuestra ANN tendrá la **arquitectura inicial**

mostrada por la Figura 12. Más adelante veremos el criterio que usaremos para la elección de estas capas ocultas.

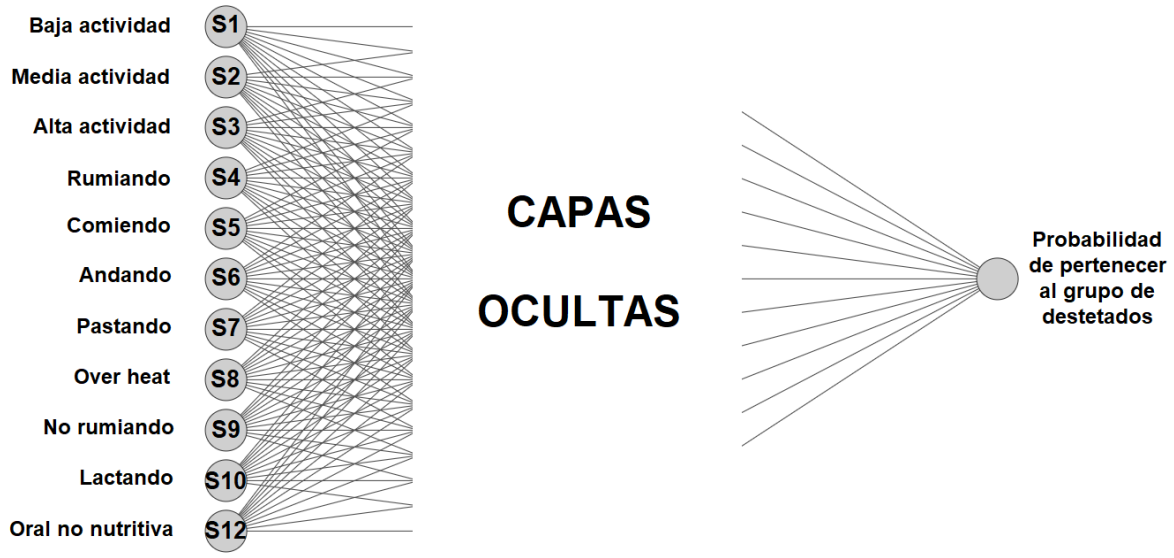


Figura 12: Arquitectura inicial de nuestra ANN.

El siguiente paso es la elección de la topología de nuestra red neuronal, dada por:

- **Funciones de activación**.

  En este caso se ha elegido la **función ReLU** para la capa de entrada y las capas ocultas, pues es la función de activación más usada en este tipo de problemas ya que no activa todas las neuronas al mismo tiempo y promueve una mayor capacidad de aprendizaje (también se ha probado con la función lineal, pero el resultado no era tan bueno). En cuanto a la función de salida se ha usado la **función Sigmoide**, pues al devolver un valor entre 0 y 1 se puede interpretar la probabilidad de pertenecer al grupo de destetados (a la hora hacer la clasificación binaria se han renombrado los números de los grupos, siendo 0 el grupo de lactantes y 1 el de destetadas). Para una explicación detallada acerca de estas funciones, ir al Apéndice A.7.2.

- **Método de optimización**.

  Al no tener una cantidad muy grande de datos, para esta red nos podemos permitir un método que converge a soluciones más óptimas, aunque el coste computacionalmente, y por tanto, el tiempo de ejecución, sea mayor. Por esta razón se ha elegido el **método del descenso del gradiente estocástico** (*SGD*) con función de coste

(función que el método trata de minimizar) el **error cuadrático medio** (*MSE*), dado por

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

siendo y_i el grupo al que pertenece el animal (0 o 1) i, y $\hat{y}_i$ la predicción en término de probabilidad (de 0 a 1). Como tasa de aprendizaje se ha usado 0.02, que es la que mejor funcionaba en este caso.

Con la topología de la red hecha, ahora tan solo nos falta terminar la arquitectura de la red y sacar los resultados. Por tanto, para encontrar la ANN que mejor se ajuste a nuestros datos hemos creado un método de optimización. Matemáticamente, este método consiste en el siguiente problema

$$\begin{aligned} \max_{ANN} \quad & ACC \\ \text{s.t.} \quad & \text{arquitectura inicial (dada por la Figura 12)}, \\ & 0 \leq \text{capas ocultas} \leq 4, \\ & \text{neuronas en las capas ocultas} \in \{0, 3, 6, \ldots, 30\}, \\ & 0 < \text{parámetros de la red} \leq \frac{\text{número de animales}}{4}, \\ & 0 < \text{epoches} \leq 500, \\ & \text{iteraciones de cross-validation} = 5. \end{aligned} \tag{4}$$

Tal y como se observa, en este problema tratamos de maximizar el accuracy de todas las posibles ANNs con estas condiciones. Veamos como hemos elegido estas condiciones:

- **Arquitectura**: Tal y como se ha explicado anteriormente, esta red tenía como base la estructura mostrada en la Figura 12. En cuanto al número de capas ocultas le hemos puesto un máximo de 4, con neuronas de 0 a 30 con saltos de 3 en 3 para limitar todas las posibles combinaciones de redes, ya que si no el tiempo de ejecución del método sería demasiado grande. Por otro lado, hemos establecido un umbral para los parámetros de la red para evitar el fenómeno de *overfitting* (ver Apéndice A.2)

- **Epoches**: en cuanto al número de entrenamientos hemos fijado el máximo en 500. Se ha elegido este número porque haciendo pruebas con varias redes llegaban a su mejor epoch entre 50 y 400, lo que significa que si se entrenaba menos el error era mayor (*underfitting*) y si se entrenaba más el error del conjunto de entrenamiento continuaba bajando pero el de test empezaba a subir (*overfitting*).

Entonces, para cada ANN que cumpla todas estas condiciones se calcula el **ACC** usando las 5 iteraciones del cross-validation de la siguiente manera:

1. Cada red se entrena 500 veces en 5 conjuntos distintos de entrenamiento y test.
2. Se saca la media del accuracy de todos los conjuntos de test para cada posible epoch.
3. El accuracy final (ACC) será el máximo de la media de los accuracy en todos los posibles epoches.

Finalmente, se selecciona la red neuronal artificial con que maximice ese ACC.

Después de aplicar este modelo a nuestro dataset, obtenemos el archivo de texto mostrado por la Tabla 5, donde se muestran los mejores epoches para todas las posibles combinaciones de arquitecturas que cumplan las condiciones impuestas en el problema 4, así como el accuracy y la desviación estándar del cross-validation resultante.

Arquitectura	**Parámetros**	**Mejor epoch**	**Accuracy**	**Std**
$[11, 1]$	12	66	73.2 %	0.044
$[11, 3, 1]$	40	128	83.3 %	0.068
$[11, 6, 1]$	79	173	85.02 %	0.065 %
⋮	⋮	⋮	⋮	

Tabla 5: Búsqueda de la mejor ANN.

Finalmente, de entre todas estas combinaciones de capas y neuronas, se elige aquella que presenta el mayor accuracy, obteniendo como resultado la siguiente arquitectura:

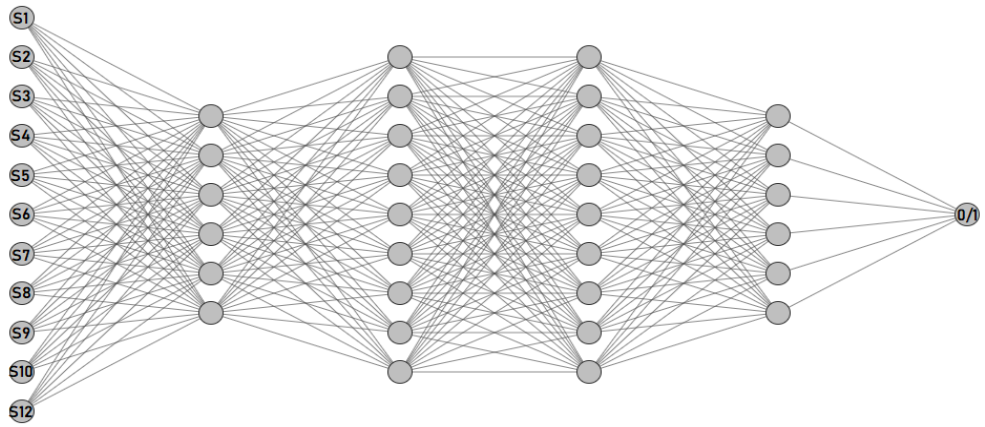


Figura 13: Arquitectura final de nuestra ANN.

Tal y como se observa en la Figura 13, esta red tiene un total de 6 capas, de las cuales 4 de ellas son ocultas. Estas capas ocultas están formadas por 6, 9, 9 y 6 neuronas, respectivamente. Por otro lado, la capa de entrada tiene 11 neuronas y la de salida 1, tal y como se había explicado anteriormente (ver Figura 12). El conjunto de estas capas forma la arquitectura de nuestra ANN, que presenta 292 parámetros para entrenar, lo cual es un número bastante bueno de parámetros, ya que es aproximadamente $\frac{1}{10}$ de las 3190 terneras con las que contamos.

Por otro lado, el número de entrenamientos elegido por el método de optimización ha sido de 124, porque, tal y como se observa en las gráficas 14, exactamente en ese epoch se consigue la máxima accuracy en los datos de test.

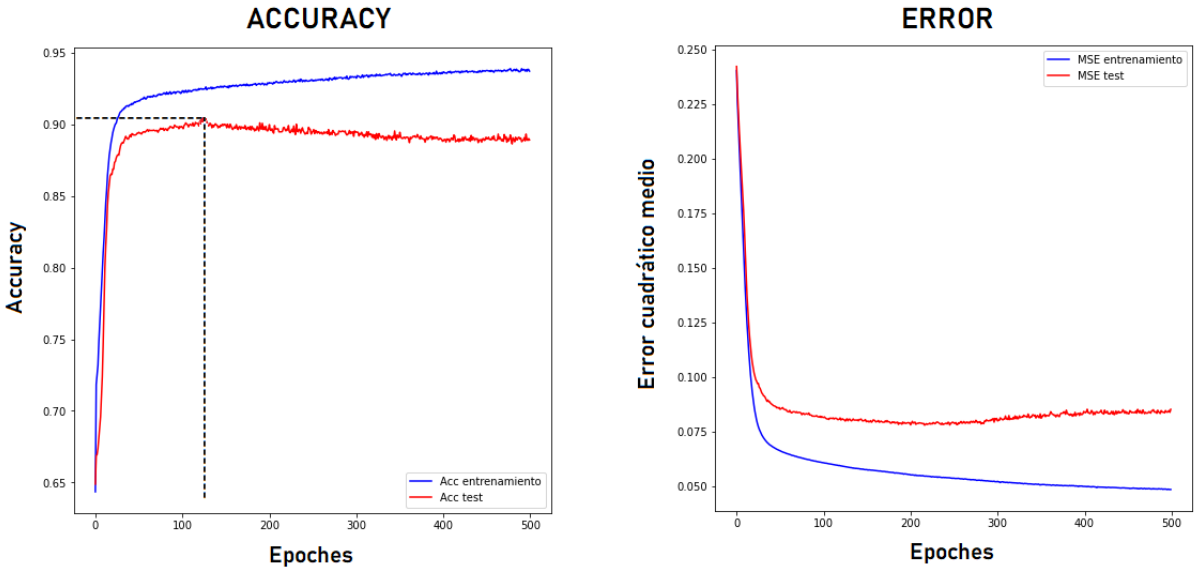


Figura 14: Gráficas del accuracy y del error cuadrático medio con nuestra ANN. En azul son los datos de entrenamiento y en rojo los del test.

De este modo, entrenando esta red con nuestros datos 124 veces para los 5 conjuntos del cross-validation, obtenemos los siguientes *accuracy*

$$[\quad 90.28\,\% \quad 97.65\,\% \quad 84.16\,\% \quad 92.48\,\% \quad 89.18\,\% \quad],$$

con *media* de 90.75 % y *desviación estándar* de 0.044.

Cada uno de los 5 modelos del cross-validation predice un grupo. Aquel grupo que haya sido predicho más veces en los 5 modelos será la predicción final. Finalmente, con este procedimiento hemos obtenido la siguiente *matriz de confusión*

		PREDICCIONES 1	PREDICCIONES 2
GRUPOS	1	1950	71
	2	177	992

con un *accuracy total* de 92.23 %. Cabe destacar que este accuracy total es el resultado de la matriz de confusión, es decir, de los 3190 animales con los que contamos, que a su vez es el resultado de la unión de los resultados de los datos de entrenamiento y de test.

Por tanto, utilizando redes neuronales artificiales hemos conseguido un modelo de clasificación con resultados bastante buenos (acierta en el 90.75 % de los casos, lo que aproximadamente es en 9 de cada 10 animales). Así que, sabiendo el grupo al cual pertenece una ternera con bastante exactitud, podemos dar paso al segundo objetivo, el predictor de actividades.

6.2. Modelo de análisis de series temporales

Para predecir las actividades que realizará una ternera perteneciente a un grupo específico vamos a usar un tipo de red neuronal recurrente llamada **LSTM**, explicada en el Apéndice A.8. En este caso, a diferencia de nuestra ANN, no vamos a hacer uso de ningún método de optimización para la búsqueda de red, pues las posibles combinaciones de redes LSTM son bastante considerables y, puesto que se tienen datos temporales de largos intervalos de tiempo, el proceso de entrenamiento de estas redes es muy lento. Por ende, nuestra red LSTM estará formada por:

- **Arquitectura**.

  En esta red se van a usar dos tipos distintos de capas, las **densas** y las **LSTM**. De este modo, para la predicción se va a usar una capa densa de entrada con el mismo número de neuronas que de minutos que se pretenden predecir, una capa densa oculta que constará de 64 neuronas, y una capa densa de salida, que tendrá el mismo número de neuronas que la de entrada. En cambio, para la información que se actualiza se va a usar una capa LSTM con 128 neuronas.

- **Funciones de activación**.

  Para las capas LSTM se ha elegido la **función Tangente Hiperbólica**, dado que es la más utilizada en este tipo de capas y la que viene por defecto. Por otro lado,

se ha elegido la **función ReLU** para la capa de entrada y las capas ocultas, pues es computacionalmente eficiente de implementar (también se ha probado con la función lineal, pero el resultado no era tan bueno). Puesto que el problema a abordar es un problema multiclase, para la función de salida se ha usado la **función Softmax** (explicada en el Apéndice A.7.2), pues esta función nos devuelve la distribución de probabilidad sobre las diferentes clases.

- **Técnica de regularización**.

  Para reducir el sobreajuste en la red, se ha usado como técnica de regularización un **dropout** ([18]) del 0.2. Esto significa que, durante el proceso de entrenamiento, se omiten aleatoriamente un 20 % de las neuronas de la capa LSTM (en nuestro caso coincide con 26 neuronas).

- **Método de optimización**.

  Al contar con una cantidad bastante grande de datos, para esta red se ha elegido un método que consiga converger más rápido que el SGT. Por esta razón se ha elegido el método de la **estimación del momento adaptativo** (*ADAM*) con un total de 3 **epoches** (con más producía *overfitting*). Por otro lado, dado que se trata de un modelo de predicción con variables categóricas, como función de perdida se ha usado de la **entropía cruzada categórica** (*categorical cross entropy*), dada por

  $$CE = -\sum_{t=1}^{T} y_t \log \hat{y}_t,$$

  siendo T el número de outputs (tiempo que queremos predecir, en minutos), y_t el estado real del animal y $\hat{y}_t$ el estado predicho.

- **Métricas**. Para los resultados de este tipo de modelos se suelen usar dos tipos de métricas, las categóricas y las probabilísticas. En este caso se ha optado por la **categórica**, dada por

  $$ACC_{cat} = \frac{1}{T}\sum_{t=1}^{T} ACC_{cat}(t),$$

  siendo

  $$ACC_{cat}(t) = \left\{ \begin{array}{lll} 0 & si & y_t \neq \hat{y}_t \\ 1 & si & y_t = \hat{y}_t \end{array} \right\}.$$

Para que la red realice buenas predicciones sin depender de la hora a la que se quiere empezar a predecir, el **método de entrenamiento** va a consistir en los siguientes pasos:

1. Se toma como input de la red los datos de las actividades de las primeras 24 horas.
2. Se entrena la red de forma que minimice la entropía cruzada categórica para las siguientes 24 horas.
3. Se repiten los pasos 1 y 2 cogiendo los datos adelantados una hora en cada iteración durante un número máximo de iteraciones o hasta que llegue al día que queremos predecir. Por ejemplo, en la segunda iteración se tomará como input de la red los datos de las actividades de las $1 - 25$ horas, y se entrenará de forma que minimice la función de pérdida para las siguientes 24 horas ($25 - 49$ horas).

En la Figura 15 se muestra un esquema gráfico de cómo funciona este método de entrenamiento.

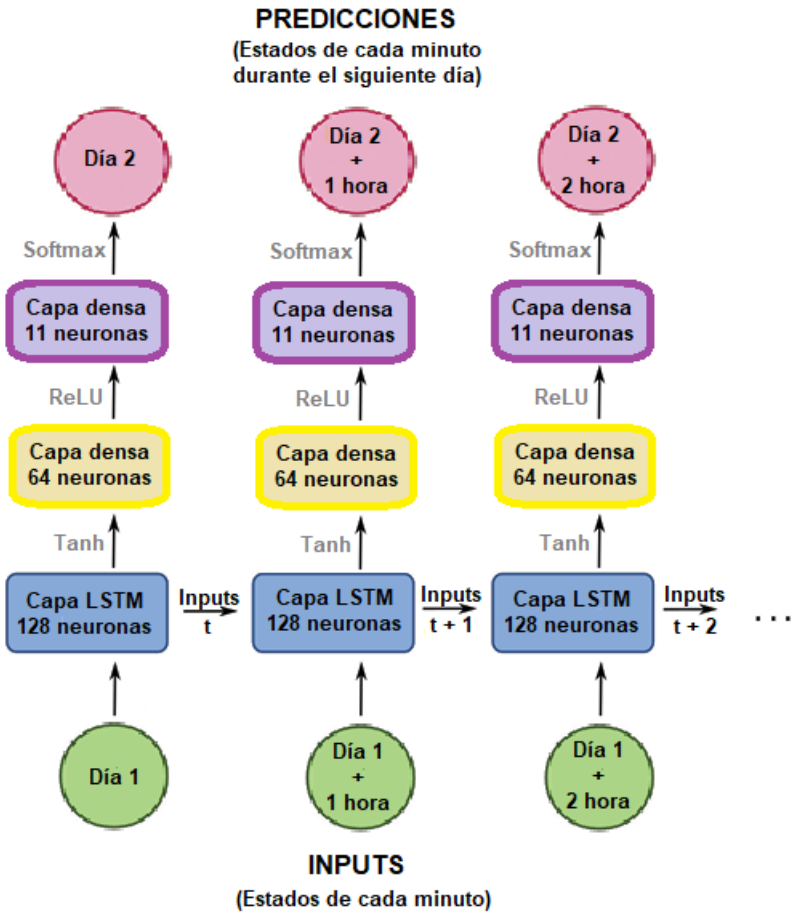


Figura 15: Método de entrenamiento de nuestra red LSTM.

Con la red ya creada, para rebajar el coste computacional, se ha escogido un total de 6 terneras para el cálculo de los resultados, 3 de ellas lactantes y las otras 3 destetadas,

de forma que estén lo más repartidas posible para evitar trabajar con datos sesgados. Por esta razón, estas terneras se han escogido de diferentes épocas del año, con la condición impuesta de que se tengan datos de como mínimo 10 días.

Nota: Todas las gráficas y resultados particulares que se van a comentar a lo largo de esta sección son de la misma ternera, en este caso, destetada.

Para esta prueba vamos a hacer uso del dataset secuencial (ver en sección 5.2), donde tenemos, para cada animal, todos los datos temporales de la actividad que está realizando durante cada minuto.

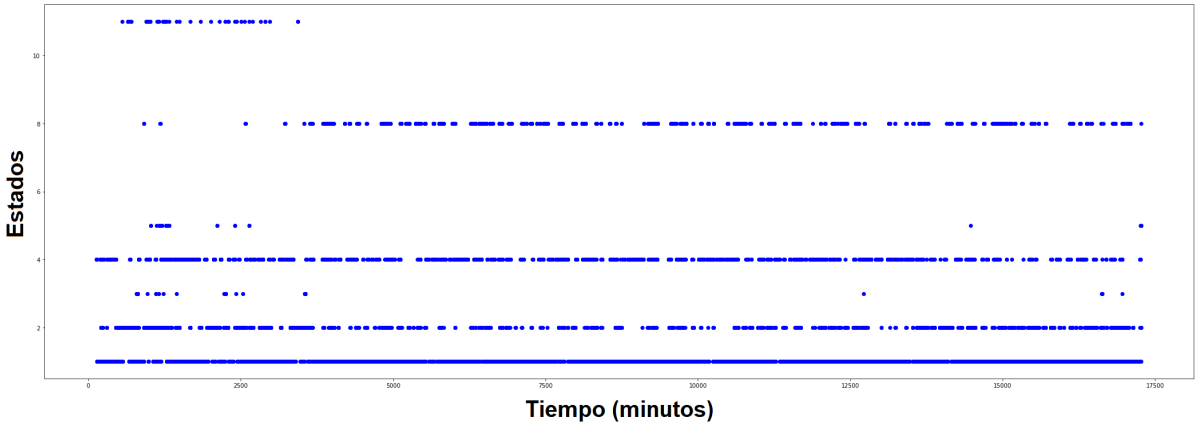


Figura 16: Estados de una ternera destetada usando el dataset secuencial.

El obejetivo principal de esta prueba es conseguir una buena predicción en un día entero de actividades. Para poder sacar resultados, se va a predecir el último día del animal donde se tengan datos registrados. En el ejemplo de la Figura 16 tenemos datos registrados en un intervalo de 12 días aproximadamente, por tanto, vamos a entrenar nuestra red con los primeros 11 días y luego predecir el último.

De este modo, entrenando nuestra red LSTM con los datos de los 6 animales, obtenemos los siguientes porcentajes de exactitud categórica

$$[\quad 24.28\,\%\quad 23.65\,\%\quad 21.16\,\%\quad 31.48\,\%\quad 25.18\,\%\quad 20.97\,\%\quad],$$

con media de 24.45 % y una desviación estándar de 3.501. En la Figura 17 se pueden observar las predicciones para la ternera ejemplo, cuyo porcentajes de exactitud categórica es del 31.48 %.

Como cabía esperar, los resultados no son buenos por varias razones. Una razón es que es muy difícil hacer una predicción con 11 posibles estados con buenos resultados de exactitud categórica, pues en predicciones continuas los errores también toman valores

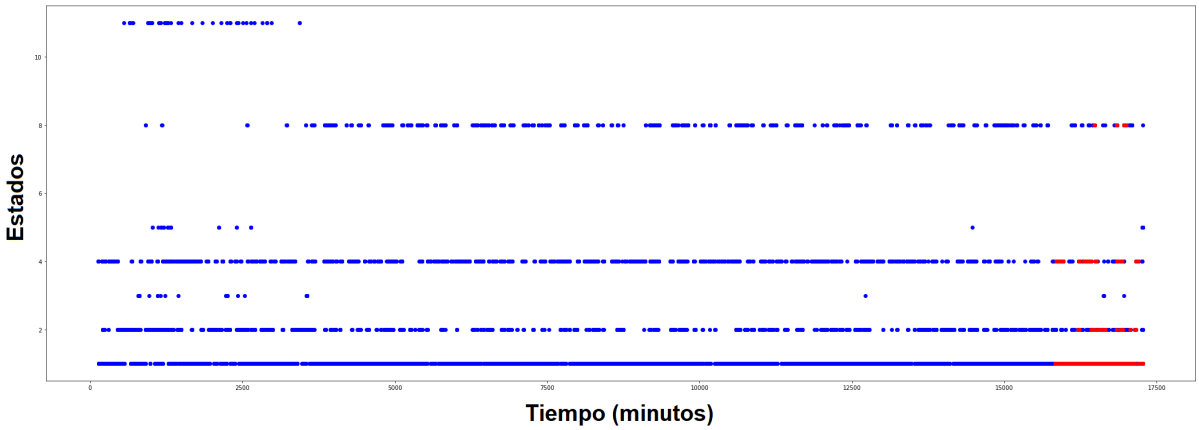


Figura 17: Predicción de los estados de un día de una ternera destetada. En azul son los estados reales, y en rojo los estados predichos.

continuos (como en el MSE), por lo que se pueden minimizar con mayor facilidad, mientras que en esta predicción estos únicamente pueden tomar el valor de acierto 0, con una probabilidad del 9 % ($\frac{1}{11}$), y el de fallo 1, con una probabilidad del 91 % ($\frac{1}{11}$).

Para mejorar esta predicción, se ha pensado en una reducción en el número de posibles estados. Para ello, se han escogido los estados de baja, media y alta actividad, de modo que haciendo la agrupación mostrada por la Figura 6 conseguimos un nuevo dataset con únicamente 3 posibles estados donde se muestra la actividad diaria de cada animal.

Estado	**Baja actividad**	**Media actividad**	**Alta actividad**
Baja actividad	×		
Media actividad		×	
Alta actividad			×
Rumiando		×	
Comiendo		×	
Andando			×
Pastando			×
Over heat			×
No rumiando	×		
Lactando		×	
Oral no nutritiva		×	

Tabla 6: Descripción de los grupos

De este modo, entrenando esta red con los nuevos datos agrupados de los 6 animales,

obtenemos los siguientes porcentajes de exactitud categórica

$$[\quad 69.55\,\%\quad 73.12\,\%\quad 65.86\,\%\quad 72.01\,\%\quad 70.45\,\%\quad 68.78\,\%\quad],$$

con media del 70.13 % y desviación estándar de 2.338. En la gráfica 18 se pueden observar las predicciones para la ternera ejemplo, cuya matriz de confusión es

		PREDICCIONES		
		Baja	Media	Alta
ACTIVIDAD	Baja	923	29	0
	Media	298	111	0
	Alta	58	18	3

con un *porcentaje de exactitud categórica* de 72.01 %.

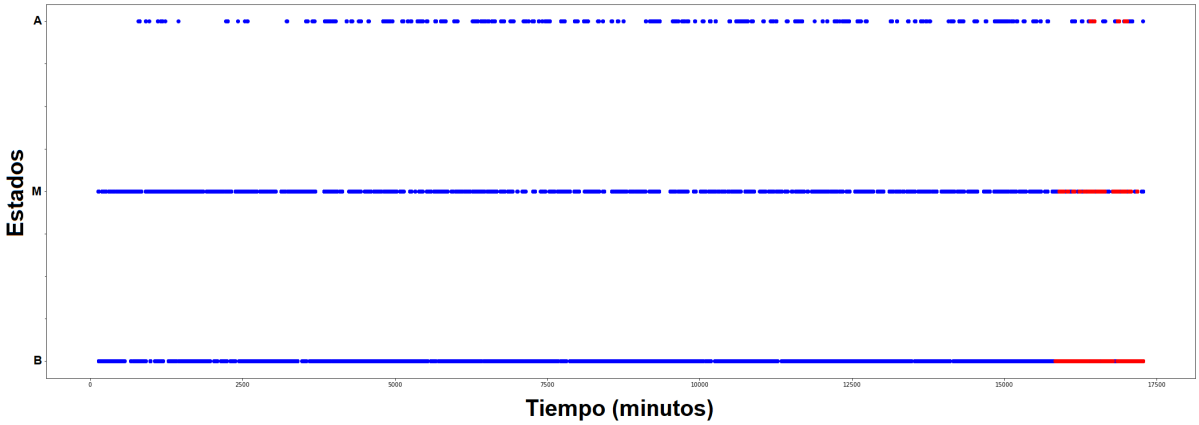


Figura 18: Predicción de los estados agrupados de un día de una ternera destetada. En azul son los estados reales, y en rojo los estados predichos.

Tal y como se puede observar, con esta agrupación los resultados son sorprendentemente prometedores. Cabe destacar que estos resultados son tan buenos (para una predicción de este tipo) dado que la mayoría de tiempo que pasan las terneras están con baja actividad (aproximadamente un 60 % del día), por lo que con una buena predicción del estado de baja actividad se puede conseguir un alto porcentaje de exactitud categórica. Por otro lado, parece que también predice medianamente bien los intervalos temporales de media actividad, en cambio, los intervalos de alta actividad parece que no se predicen adecuadamente. Esto puede ser debido a que estos animales pasan muy poco tiempo al día con estado de alta actividad, lo que hace que la red no tenga mucho en cuenta este estado.

7. Conclusiones y trabajo futuro

Durante el presente proyecto, se ha abordado la aplicación de técnicas de aprendizaje automático en el contexto de terneras pertenecientes a granjas no extensivas. Se han presentado y analizado desde una perspectiva puramente teórica diversos algoritmos y enfoques del machine learning que son particularmente adecuados para abordar los objetivos específicos con los datos proporcionados por los proyectos *Re-livestock* y *TED-Farm*.

Para abordar el objetivo principal de este trabajo, es decir, la clasificación de terneras en diferentes grupos alimenticios según sus actividades diarias, se intentó hacer una agrupación mediante clústeres. Como resultado del clustering K-means, se obtuvo un clúster que clasifica terneras destetadas con un 96.06 % de acierto, mientras que en los otros dos se entremezclaban los grupos. Este resultado nos indica que, de alguna manera, algunas terneras destetadas se comportan de distinta manera a las lactantes (de 2 o 3 tomas). Usando los centroides de los clústeres (ver figura 9), se puede observar que el clúster que ha conseguido clasificar las terneras destetadas tiene un animal centroide bastante activo, mientras que los animales centroides de los otros clústeres están más inactivos. Con estos resultados se puede afirmar que, como norma general, las terneras destetadas son más activas que las lactantes.

Por otro lado, hay varias terneras destetadas que no pertenecen a este clúster. Para entender el comportamiento de estos animales, se graficaron (Figura 10) los tres estados de actividad (baja, media y alta) de todos los terneros con su respectivo grupo. En esta gráfica se muestra lo mencionado anteriormente, y es que las terneras más activas son casi todas destetadas, en cambio, las terneras de 2 y 3 tomas no se diferencian prácticamente en cuanto al estado de la actividad. También se puede observar que hay un conjunto bastante grande de terneras destetadas que no están con alta actividad en casi ningún momento. En este caso, el 65 % de estas terneras han pasado por un estado de sobrecalentamiento (*over-heat*) en un largo período del día, pero el 35 % de terneras restantes no han pasado por ningún estado de sobrecalentamiento a lo largo del día. Esto puede ser por varias razones, entre ellas que se hayan puesto enfermas al terminar con la lactancia.

Como conclusión, estos resultados muestran que, únicamente con los estados de actividad y de sobrecalentamiento de las terneras destetadas, se pueden detectar aquellas terneras que podrían estar pasando por una enfermedad.

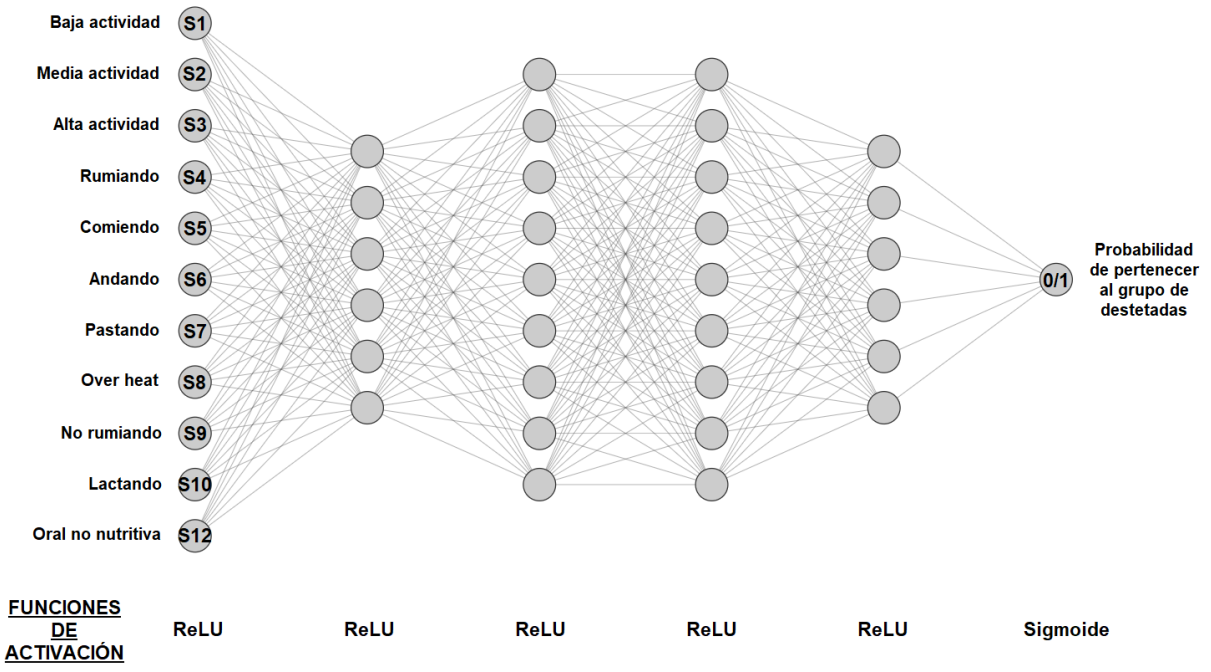
A pesar de este asombroso resultado, parece ser que mediante clustering (aprendizaje no supervisado) no se consiguió una buena clasificación de los tres grupos alimenticios (41.32 % de accuracy). Por esta razón, aprovechando que se conocen los grupos de cada

animal, se probó con un SVM con kernel polinomial de grado 2 (el kernel con mejores resultados). Con este nuevo método de aprendizaje supervisado se consiguió un accuracy total del 63.54 %, que sigue siendo un resultado poco satisfactorio.

Estos resultados sugieren que es probable que los grupos lactantes tengan un comportamiento similar, ya sea que estén con 2 o con 3 tomas diarias. Por esta razón, se consideraron los grupos de 2 y 3 tomas como un único grupo.

Esta nueva clasificación nos permitió la utilización de la regresión logística binaria como método de clasificación, de modo que, haciendo pruebas y usando la matriz de correlaciones (Figura 11), se concluyó que la función logística que más se adapta a nuestros datos es la dada por la ecuación 3, con un accuracy del 75 %.

Dado que con modelos de machine learning tradicional no se consiguió un clasificador con buenos resultados, se decidió probar con redes neuronales artificiales. De este modo, mediante pruebas y un método de optimización creado para la búsqueda de la mejor arquitectura, se concluyó que la ANN que mejor se adapta a nuestros datos es la siguiente:



Finalmente, usando el método del descenso del gradiente estocástico como optimizador de esta red se consiguió, después de 124 entrenamientos, un accuracy del 90.75 %, lo cual es un resultado prometedor.

En definitiva, la nueva clasificación con los grupos de lactantes y destetadas se ha cumplido con resultados muy buenos (90.75 % de accuracy) y, además, se han logrado

conclusiones con consecuencias directas en granjas, como la detección de terneras destetadas enfermas.

En último lugar, para abordar el segundo y último objetivo del trabajo, es decir, la predicción de las actividades que realizará en un día entero una ternera perteneciente a un grupo específico, se probó con un tipo de redes neuronales recurrentes cuyo objetivo particular es el análisis de series temporales con datos discretos.

La red que se eligió fue una LSTM, con una capa de entrada de 1440 neuronas (los minutos que tiene 1 día, el tiempo que se quiere predecir), una capa oculta LSTM de 128 neuronas, una capa oculta densa de 64 y una capa de salida de 1440 neuronas (igual que la de entrada). En cuanto las funciones de activación, tanto en la capa de entrada como en la densa oculta se eligió la función $ReLU$, y luego para la capa LSTM se eligió la función tanh y para la de salida la *softmax*.

De este modo, usando el método de la estimación del momento adaptativo como optimizador de esta red, se consiguió, después de 3 entrenamientos para cada uno de los 6 animales elegidos de manera que se evite trabajar con datos sesgados, un porcentaje de exactitud categórica del 24.45 % que, como cabía esperar, es un resultado bastante malo.

Una razón de estos resultados tan pésimos es que es muy difícil hacer una predicción con 11 posibles estados con buenos resultados de exactitud categórica, pues en este caso los errores únicamente pueden tomar el valor de acierto 0 (con una probabilidad del 9 %) y el de fallo 1 (con una probabilidad del 91 %). Con el fin de abordar esta problemática, se decidió reducir el número de estados y agruparlos en tres niveles de actividad (baja, media y alta) utilizando los criterios establecidos en la Tabla 6.

Finalmente, con esta nueva agrupación y la red LSTM, se consiguió un porcentaje de exactitud categórica del 70.13 % que, para un problema de predicción discreta, es un resultado bastante prometedor.

Como conclusión, estos resultados muestran que, con los datos temporales de baja, media y alta actividad de una ternera y una red LSTM, se pueden predecir razonablemente las actividades realizadas por una ternera a lo largo de un día. Una consecuencia directa de esta predicción podría ser la detección de terneras enfermas, pues si la predicción muestra un comportamiento muy distinto al que está realizando una ternera, podría ser un indicador de que esa ternera está enferma. Por ejemplo, si se predice que una ternera a lo largo del día tendría que estar activa y no presenta casi estados de media y alta actividad, o viceversa.

En el trabajo futuro, se busca una mejora significativa en los resultados de predicción obtenidos hasta ahora. Para lograrlo, se planea llevar a cabo una exploración más exhaustiva y una evaluación más rigurosa de diferentes opciones de redes neuronales. Esto incluirá probar diversas arquitecturas de redes LSTM, ajustando sus parámetros y analizando su impacto en la predicción de las actividades diarias de las terneras. Además, se considerará la incorporación de otro tipo de redes neuronales recurrentes, como las GRU (Gated Recurrent Unit), para comparar su desempeño y determinar cuál se adapta mejor al problema en cuestión.

Además de la mejora en la predicción de actividades, se tiene como objetivo el desarrollo de un modelo más sofisticado que vaya más allá de la simple predicción. Este modelo buscará detectar terneras enfermas utilizando intervalos de confianza. Al establecer límites de confianza en las predicciones, se podrá identificar de manera más precisa y fiable cualquier comportamiento atípico que pueda indicar un problema de salud en las terneras. Esta capacidad de detección temprana y precisa permitirá una respuesta más oportuna y efectiva en términos de cuidado y tratamiento veterinario.

En conjunto, este enfoque ampliado, que implica explorar diferentes redes neuronales y desarrollar un modelo con intervalos de confianza, tiene como objetivo mejorar tanto la precisión en la predicción de las actividades diarias de las terneras como la capacidad de detección de enfermedades en el manejo de ganado. Esto no solo contribuirá a optimizar el bienestar de las terneras, sino que también brindará a los productores ganaderos herramientas más efectivas para tomar decisiones informadas y garantizar una gestión más eficiente y saludable de su ganado.

Referencias

[1] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[2] Anup Bhattacharya, Jan Eube, Heiko Röglin, and Melanie Schmidt. Noisy, greedy and not so greedy k-means++. *arXiv preprint arXiv:1912.00653*, 2019.

[3] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[4] María Pilar González Casimiro. Análisis de series temporales: Modelos arima. 2009.

[5] Gus W Haggstrom. Logistic regression and discriminant analysis by ordinary least squares. *Journal of Business & Economic Statistics*, 1(3):229–238, 1983.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[7] Mutasem Khalil Alsmadi, Khairuddin Bin Omar, Shahrul Azman Noah, and Ibrahim Almarashdah. Performance comparison of multi-layer perceptron (back propagation, delta rule and perceptron) algorithms in neural networks. In *2009 IEEE International Advance Computing Conference*, pages 296–299. IEEE, 2009.

[8] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[9] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[10] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[11] Gianluca Pollastri, Darisz Przybylski, Burkhard Rost, and Pierre Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002.

[12] Tony Robinson, Mike Hochberg, and Steve Renals. The use of recurrent neural networks in continuous speech recognition. *Automatic speech and speaker recognition*, pages 233–258, 1996.

[13] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[14] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[15] Sergi Sanjuan Silvestre. Problemas de clasificación y predicción en ganadería de precisión. `https://github.com/serjj99/PrecisionLivestockFarming`, 2023. [Online; acceso el 10/07/2023].

[16] Roy Nuary Singarimbun, Erna Budhiarti Nababan, and Opim Salim Sitompul. Adaptive moment estimation to minimize square error in backpropagation algorithm. In *2019 International Conference of Computer Science and Information Technology (ICoSNIKOM)*, pages 1–7. IEEE, 2019.

[17] KP Soman, R Loganathan, and V Ajay. *Machine learning with SVM and other kernel methods*. PHI Learning Pvt. Ltd., 2009.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

A. Modelos matemáticos desarrollados

A.1. Cross-validation

La **validación cruzada**, o *cross-validation* en inglés, es una técnica comúnmente utilizada en el machine learning para evaluar el rendimiento de un modelo estadístico o algoritmo de aprendizaje. Su objetivo principal es estimar qué tan bien se generaliza un modelo a datos no vistos.

Existen varios enfoques de esta técnica, pero la más utilizada, así como la que se utiliza en este trabajo, es la **validación cruzada de K iteraciones**. Este tipo de cross-validation, también llamada *K-fold cross-validation*, se basa en dividir los datos disponibles en K conjuntos o "folds" (pliegues). El procedimiento típico de este método implica los siguientes pasos:

1. **División de los datos**: El conjunto de datos se divide en K conjuntos, con K un número predefinido (hiperparámetro). Cada conjunto se elige de manera aleatoria y es aproximadamente del mismo tamaño.

2. **Iteración del modelo**: Se ejecuta el modelo K veces. En cada iteración, se selecciona un pliegue diferente como conjunto de prueba y los restantes se utilizan como conjunto de entrenamiento.

3. **Entrenamiento y evaluación del modelo**: En cada iteración, el modelo se entrena utilizando el conjunto de entrenamiento y se evalúa utilizando el conjunto de prueba. Se registra una métrica de evaluación, como el error medio cuadrático (*mse*) o la precisión (*accuracy*), para cada iteración.

4. **Promedio de los resultados**: Una vez completadas las K iteraciones, se calcula el promedio de las métricas de evaluación obtenidas en cada iteración. Esto proporciona una estimación más robusta y fiable del rendimiento del modelo.

Esta validación cruzada tiene varias ventajas. En primer lugar, aprovecha al máximo los datos disponibles al utilizar diferentes combinaciones de entrenamiento y prueba, lo cual es una gran ventaja cuando se trabaja con pocos datos. Además, proporciona una estimación más precisa y menos sesgada del rendimiento del modelo en comparación con una única división de los datos en un conjunto de entrenamiento y un conjunto de prueba. También es útil para seleccionar hiperparámetros óptimos, ya que se puede utilizar para comparar diferentes configuraciones del modelo.

En resumen, la validación cruzada es una técnica fundamental en el aprendizaje automático cuando se trabaja con pocos datos, pues nos permite evaluar y comparar modelos de manera más confiable, puesto que evita el sesgo a la hora de elegir los datos de entrenamiento o los de prueba.

A.2. Overfitting

El **sobreajuste de los datos**, o *overfitting* en inglés, es un fenómeno común en el machine learning que ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y tiene dificultades para generalizar correctamente a nuevos datos.

El overfitting ocurre cuando un modelo se vuelve demasiado complejo o flexible y "memoriza" los detalles y el ruido presentes en los datos de entrenamiento en lugar de capturar las relaciones y patrones subyacentes que son generalizables. Como resultado, el modelo se ajusta perfectamente a los datos de entrenamiento, pero su rendimiento se degrada significativamente cuando se le presentan nuevos datos, tal y como se observa en la Figura 19.

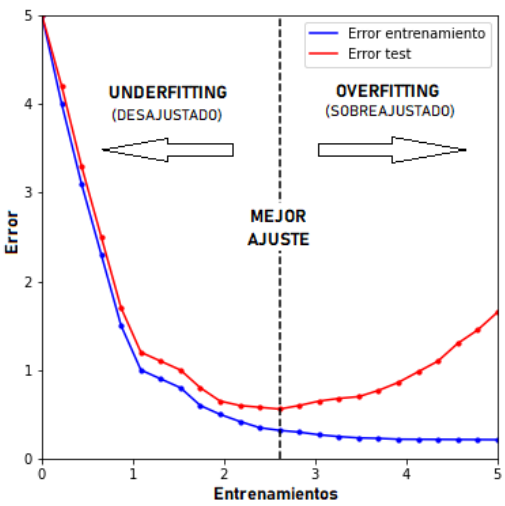


Figura 19: Ejemplo de underfitting y overfitting.

Hay varias razones por las que el overfitting puede ocurrir:

- **Sobreentrenamiento**: Esto ocurre cuando un modelo se entrena durante demasiadas iteraciones o épocas, lo que lleva a un ajuste excesivo a los datos de entrenamiento.

- **Alta complejidad del modelo**: Si se utiliza un modelo muy complejo con una gran capacidad de aprendizaje, como una red neuronal con muchos parámetros, existe un mayor riesgo de overfitting. Estos modelos pueden aprender incluso el ruido y las fluctuaciones aleatorias presentes en los datos de entrenamiento.

- **Datos insuficientes**: Cuando se dispone de un conjunto de datos pequeño, el modelo puede encontrar dificultades para aprender patrones genuinos y, en cambio, se

ajusta a los datos de manera excesiva. En tales casos, el modelo puede generalizar mal a nuevos datos debido a la falta de diversidad en el conjunto de entrenamiento.

Para evitar este problema, en este trabajo se ha trabajado con algoritmos creados desde cero para encontrar el mejor número de iteraciones de entrenamiento o el mejor número de parámetros de una red neuronal sin que se sobreajusten los datos.

A.3. Agrupamiento jerárquico

El agrupamiento jerárquico (clustering jerárquico) es un método de agrupamiento de objetos en conjuntos basado en la **conectividad**. En este enfoque, se construye una estructura jerárquica de clústeres, donde los objetos se agrupan de forma recursiva y se forman subgrupos dentro de grupos más grandes dependiendo de la similitud entre estos.

Para comenzar, consideremos un conjunto de datos con n objetos que queremos agrupar. Representemos cada objeto mediante un vector de p variables explicativas, denotado como $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p})$, donde i representa el índice del objeto.

El primer paso del clustering jerárquico implica calcular una matriz de distancias o similitudes entre los objetos. Esta matriz, llamada **matriz de similitud** o **matriz de distancias**, mide la proximidad entre cada par de objetos del conjunto de datos. La elección de la **medida de similitud** o **distancia** depende del problema y de la naturaleza de los datos. Algunas de las medidas más comunes vienen dadas por la Tabla 7.

Ahora que sabemos como calcular las distancias entre dos pares de puntos, ¿cómo calculamos las distancias entre los conjuntos de observaciones, en lugar de observaciones individuales? Para responder a esta pregunta, es necesario comprender qué son y para qué se utilizan los métodos de enlace.

Los **métodos de enlace**, también conocidos como *linkage methods*, son reglas o medidas utilizadas en el clustering jerárquico para determinar la distancia o similitud entre dos grupos al fusionarlos en cada etapa del proceso. Estos métodos definen cómo se calcula la distancia entre dos grupos a partir de las distancias entre sus elementos individuales.

Entre los métodos de enlace, los más comunes en el clustering jerárquico son:

- **Enlace único** (*Single-linkage*): También conocido como *enlace de vecino más cercano*, este método calcula la distancia entre dos grupos como la distancia mínima entre cualquier par de elementos de los dos grupos. Matemáticamente, dados dos grupos A y B y un par de elementos $x_i \in A$ y $x_j \in B$, se calcula la distancia entre

Distancia	**Fórmula**	**Contexto**
Manhattan (l_1)	$d_1(x_i, x_j) = \sum_{k=1}^{p} \lvert x_{i,k} - x_{j,k} \rvert$	Cuando se trabaja con variables discretas o categóricas
Euclideana (l_2)	$d_2(x_i, x_j) = \sqrt{\sum_{k=1}^{p} (x_{i,k} - x_{j,k})^2}$	Cuando las variables tienen una escala similar y la geometría euclidiana es relevante
Canberra	$d_C(x_i, x_j) = \sum_{k=1}^{p} \frac{\lvert x_{i,k} - x_{j,k} \rvert}{\lvert x_{i,k} \rvert + \lvert x_{j,k} \rvert}$	Cuando las variables tienen escalas diferentes y la relación proporcional entre ellas es más importante que las diferencias absolutas.

Tabla 7: Medidas más comunes.

los dos grupos como

$$D_s(A, B) = \min(d(x_i, x_j)).$$

- **Enlace completo** (*Complete-linkage*): También conocido como *enlace de vecino más lejano*, este método calcula la distancia entre dos grupos como la distancia máxima entre cualquier par de elementos de los dos grupos. En otras palabras, dados dos grupos A y B y un par de elementos $x_i \in A$ y $x_j \in B$, se calcula la distancia entre los dos grupos como

$$D_c(A, B) = \max(d(x_i, x_j)).$$

- **Enlace promedio** (*Average-linkage*): Este método calcula la distancia entre dos grupos como el promedio de las distancias entre todos los pares de elementos de los dos grupos. Proporciona una medida más equilibrada que el enlace único y el enlace completo, ya que considera todas las distancias en lugar de solo las más cercanas o más lejanas. Matemáticamente, dados dos grupos A y B y un par de elementos $x_i \in A$ y $x_j \in B$, este método calcula la distancia entre los dos grupos como

$$D_a(A, B) = \frac{\sum_{i,j} d(x_i, x_j)}{|A| \cdot |B|},$$

donde $|A|$ y $|B|$ son las cardinalidades (número de elementos) de los grupos A y B, respectivamente.

- **Método Ward** (*Ward-linkage*): También conocido como *método de varianza mínima*, este método compara la suma de errores cuadrados dentro de los clústeres con la suma de los cuadrados dentro de la unión de los conglomerados, fusionándolos para minimizarlos. Para clústeres disjuntos C_i, C_j y C_k, este método puede ser implementado por la *fórmula de Lance–Williams*

$$d_w(C_i \cup C_j, C_k) = \frac{|C_i| + |C_k|}{|C_i| + |C_j| + |C_k|} d_w(C_i, C_k) + \\ + \frac{|C_i| + |C_j|}{|C_i| + |C_j| + |C_k|} d_w(C_i, C_j) + \frac{|C_j| + |C_k|}{|C_i| + |C_j| + |C_k|} d_w(C_j, C_k).$$

  Este elaborado método es altamente eficaz en el análisis de clustering jerárquico, y es tan ampliamente reconocido que se encuentra implementado por defecto en la mayoría de las librerías y herramientas que ofrecen funcionalidades de clustering jerárquico. Su popularidad se debe a su capacidad para producir resultados robustos y de alta calidad en diversos conjuntos de datos.

  Uno de los principales beneficios del método Ward es que tiende a producir grupos de tamaño similar y compactos. Además, es especialmente útil cuando los datos contienen valores numéricos y la varianza es importante para la interpretación del problema.

Estos son solo algunos de los métodos más comunes utilizados en el clustering jerárquico. Cada uno tiene sus propias ventajas y desventajas, y la elección del método depende del tipo de datos y los objetivos del análisis de clustering.

Una vez que tenemos la matriz de similitud o distancia y el método de enlace, podemos iniciar el **proceso de agrupamiento**. Este proceso puede ser de dos tipos:

- **Aglomerativo**: En este enfoque comenzamos con n clústeres, cada uno representando un objeto individual. Luego, iterativamente, fusionamos los dos clústeres más cercanos en función de la distancia entre ellos. Este proceso se repite hasta que todos los objetos se agrupen en un único clúster que contiene todos los elementos.

- **Divisivo**: En este enfoque comenzamos con un único clúster que contiene todos los objetos. Luego, iterativamente, dividimos el clúster actual en subgrupos más pequeños en función de la similitud entre los objetos. Este proceso se repite hasta que cada objeto se encuentre en su propio clúster individual.

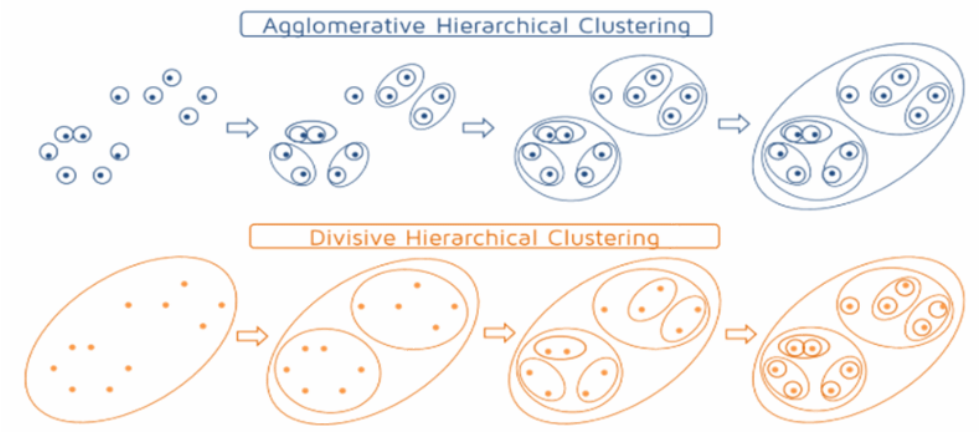


Figura 20: Agrupamiento aglomerativo vs divisivo.

Una vez finalizado el proceso de clustering jerárquico, obtenemos una jerarquía de clústeres que refleja la similitud entre los objetos. Podemos visualizar esta jerarquía utilizando el **dendrograma**, que muestra la estructura jerárquica y las fusiones/divisiones realizadas en cada paso. En la Figura 21 se puede observar un ejemplo de este diagrama de árbol, donde el eje x representa los objetos y el eje y la similitud o distancia entre estos.

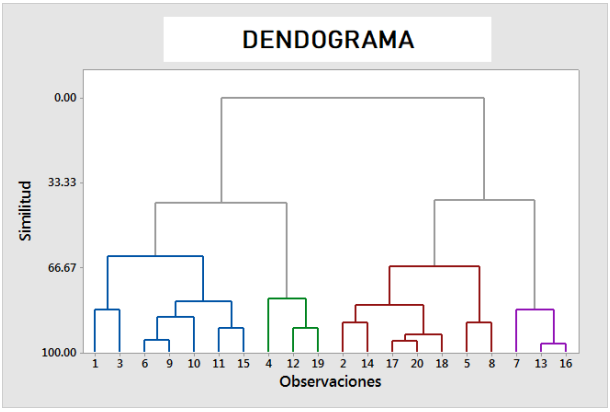


Figura 21: Ejemplo de un dendograma.

En resumen, el clustering jerárquico es un método matemático para agrupar objetos en conjuntos basados en sus distancias, mediante el cálculo de una matriz de similitud o distancia y la construcción de una estructura jerárquica de clústeres.

A.4. K-means

El algoritmo k-medias, en inglés conocido como *k-means*, es el algoritmo de aprendizaje automático no supervisado más utilizado para dividir un conjunto de datos dado en k clústeres. Este método se basa en el **centroide**, es decir, en puntos representativos dentro de un conjunto de datos. En este contexto, k representa el número de grupos o clústeres y debe ser fijado previamente (hiperparámetro).

La idea básica que subyace a la agrupación de k-medias consiste en definir los clústeres de forma que se minimice la variación total intra-clúster para el conjunto de los k clústeres. De este modo, si tenemos k clústeres sobre un conjunto de datos $C = \{C_1, C_2, \ldots, C_k\}$ el algoritmo se basa en

$$\min_C E(\mu_i) = \min_C \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_j\|^2,$$

donde μ_i es el centroide de cada cúster.

A grandes rasgos, el algoritmo k-medias sigue los siguientes pasos:

1. **Inicialización**: Se especifican el número de clústeres k que deseamos y se seleccionan aleatoriamente k puntos $x_i, i = 1, \ldots, k$ como centroides iniciales o se definen de alguna otra manera. Estos centroides representan los puntos centrales de cada clúster.

2. **Asignación de puntos a clústeres**: Cada punto de datos se asigna al clúster cuyo centroide es el más cercano en términos de la distancia previamente elegida, que suele ser la euclídea.

3. **Actualización de centroides**: Se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo. Esto implica mover el centroide hacia el centro de los puntos asignados.

4. **Repetición de los pasos** 2 **y** 3: Los pasos 2 y 3 se repiten iterativamente hasta que los centroides converjan y no haya cambios significativos en las asignaciones de puntos a clústeres o hasta que alcance un número máximo de iteraciones.

El resultado final es un conjunto de k clústeres donde los puntos dentro de cada clúster son similares entre sí y distintos de los puntos en otros clústeres.

En cada actualización de los centroides, imponemos la condición de extremo para la función $E(\mu_i)$

$$\frac{\partial E(\mu_i)}{\partial \mu_i} = 0. \tag{5}$$

Para ver esa derivada vamos a reescribir la ecuación anterior en forma matricial, de esta forma nos queda

$$E(\mu_i) = \sum_{i=1}^{k} \sum_{x_j \in C_i} (x_j - \mu_i)^T I (x_j - \mu_i),$$

donde I es la matriz identidad y T significa transpuesta.

Entonces, utilizando la condición de extremo podemos deducir la posición de los centroides. Para ello, si tenemos que

$$B = x^T A x,$$

donde A es una matriz simétrica y tomamos $\frac{\partial}{\partial x}$ como el vector gradiente en forma de columna, llegamos a la condición

$$\frac{\partial B}{\partial x} = Ax + A^T x. \tag{6}$$

Finalmente, aplicando la condición (5) y la ecuación (6) a la función $E(\mu_i)$, nos queda

$$\begin{aligned}
\frac{\partial E(\mu_i)}{\partial \mu_i} = \sum_{x_j \in C_i} I(x_j - \mu_i) + I^T(x_j - \mu_i) = 0 \\
\sum_{x_j \in C_i} (x_j - \mu_i) = 0 \\
\sum_{x_j \in C_i} x_j - \sum_{x_j \in C_i} \mu_i = \sum_{x_j \in C_i} x_j - |C_i|\mu_i = 0 \\
\implies \mu_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j.
\end{aligned}$$

Este algoritmo se basa en el principio de minimización de la inercia, que busca minimizar la suma de las distancias al cuadrado dentro de cada clúster. Sin embargo, el resultado final puede depender de la inicialización de los centroides, por lo que a menudo se ejecuta varias veces con diferentes inicializaciones y se selecciona la solución con la menor inercia.

En resumen, el algoritmo k-means es un método matemático utilizado para agrupar datos en clústeres, donde se busca minimizar las distancias entre los puntos y los centroides. Es una técnica ampliamente utilizada en análisis de datos y aprendizaje no supervisado.

Existen otros algoritmos de clustering, como los que se basan en la distribución o en la densidad. Un ejemplo de este último es el **DBSCAN** (*Density-Based Spatial clustering of Applications with Noise*), un algoritmo de agrupamiento bastante común que no se desarrollará ni aplicará en este trabajo.

Para una comprensión más completa y detallada sobre el clustering jerárquico, recomendamos el artículo [8]. Este artículo proporciona una lectura en profundidad sobre el tema, junto con numerosos ejemplos que ilustran el funcionamiento y la aplicación de estos algoritmos.

A.5. Regresión logística

La **regresión logística** es un **método estadístico supervisado** utilizado para modelar y predecir la relación entre una variable de respuesta categórica y un conjunto de variables predictoras. El objetivo es estimar la probabilidad de que una observación pertenezca a una de las categorías en función de los valores de las variables predictoras. Este tipo de regresión utiliza una función logística para transformar una combinación lineal de las variables predictoras en una probabilidad en el rango de 0 a 1. Luego, se utiliza un umbral para clasificar la observación en una de las categorías en función de su probabilidad estimada.

En este trabajo únicamente se utiliza para **clasificación binaria**. Es por ello que la posterior explicación de este método se centra en que únicamente hay dos categorías.

Las regresiones logísticas modelan la relación entre la variable de respuesta categórica y las covariables. Específicamente, existe una combinación lineal de variables independientes con el logaritmo de la razón de probabilidad de un evento en un modelo logístico. Las regresiones logísticas binarias estiman la probabilidad de que una característica de una variable binaria esté presente, dadas los valores de las covariables. Supongamos que Y es una variable de respuesta binaria, donde $Y_i = 1$ si la característica está presente y $Y_i = 0$ si la característica está ausente, y los datos $[Y_1, Y_2, \ldots, Y_n]$ son independientes. Además, consideremos $x = (x_1, x_2, \ldots, x_p)$ como un conjunto de variables explicativas que pueden ser discretas, continuas o una combinación de ambas. Sea π_i la probabilidad de éxito, entonces la función logística para π_i está dada por

$$Logit(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p},$$

donde

$$\pi_i = \frac{\exp(\beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p})}{1 + \exp(\beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p})} = \frac{\exp(x_i'\beta)}{1 + \exp(x_i'\beta)}.$$

Aquí, π_i denota la probabilidad de que una muestra esté en una categoría específica de la variable de respuesta binaria, comúnmente conocida como "probabilidad de éxito", y claramente, $0 \leq \pi_i \leq 1$. Por otro lado, β representa un vector de parámetros a ser estimados.

Aunque se puede utilizar el enfoque de mínimos cuadrados (ver [5]), el **método de máxima verosimilitud** es el preferido para estimar β, ya que tiene mejores propiedades estadísticas. Para ello consideremos el modelo logístico con una única variable predictora

X dado por la función logística

$$\pi(X) = \frac{exp(X_i\beta)}{1 + \exp(X_i\beta)}.$$

Deseamos encontrar las estimaciones para que, al sustituir $\hat{\beta}$ en el modelo para $\pi(X)$, obtengamos un número cercano a uno para todos los sujetos que tienen diabetes y cercano a cero para los demás. Matemáticamente, la función de verosimilitud viene dada por

$$L(\beta) = \prod_{i:y_i=1} \pi(x_i) \prod_{i':y_{i'}=0} (1 - \pi(x_{i'})).$$

Las estimaciones $\hat{\beta}$ se eligen maximizando esta función de verosimilitud.

Otra característica de este método es la penalización utilizada, con el fin que intentar reducir los pesos del modelo. Existen varios tipos de penalizaciones, pero en este trabajo únicamente se usará la **penalización** L_2, pues es la más común. En este caso, la **función de verosimilitud** viene dada por

$$L(\beta) = \prod_{i:y_i=1} \pi(x_i) \prod_{i':y_{i'}=0} (1 - \pi(x_{i'})) - C \sum_{i=1}^{p} w_i^2, \tag{7}$$

siendo C la constante de penalización (hiperparámetro) y w_i los pesos de cada variable. Cuanto más grande sea C más penalización de pesos se produce y viceversa, tal y como indica la ecuación 7.

A.6. SVM

Las **Máquinas de Vectores de Soporte** (SVM, de sus siglas en inglés "*Support Vector Machine*") es un **algoritmo de aprendizaje supervisado** utilizado para resolver problemas de clasificación y regresión. Su objetivo principal es encontrar un hiperplano en un espacio dimensional superior que maximice la separación entre diferentes clases de puntos de datos.

En un contexto de clasificación, supongamos que tenemos un conjunto de puntos de datos que queremos clasificar en dos clases diferentes. El SVM busca encontrar un hiperplano que pueda separar de manera óptima los datos en dos clases diferentes. Matemáticamente, el hiperplano puede ser representado como:

$$w^T x + b = 0,$$

donde w es un vector de pesos que define la dirección y orientación del hiperplano, x es un vector de características de entrada y b es un sesgo (bias). La función de decisión se define como

$$f(x) = w^T x + b.$$

El objetivo del SVM es encontrar los valores óptimos de w y b que maximicen el margen entre el hiperplano y los vectores de soporte, que son los puntos más cercanos al hiperplano de cada clase.

En cuanto a la capacidad del SVM para lidiar con conjuntos de datos que pueden no ser completamente linealmente separables, existen dos tipos de modelos:

- **Con margen duro** (*Hard-Margin SVM*)

  En este caso se asume que los datos de entrenamiento son **linealmente separables**, es decir, existe un hiperplano que puede separar perfectamente las diferentes clases sin errores. El objetivo del hard-margin SVM es encontrar ese hiperplano de manera que los puntos de datos de diferentes clases queden claramente separados.

  Sea $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ un conjunto de datos donde y_i es la etiqueta de clase a la que pertenece el punto x_i (1 para la primera clase y -1 para la segunda clase) y N es el número total de puntos de datos, entonces este modelo se puede expresar como el siguiente problema de optimización convexa

  $$\min \ \frac{1}{2}\|w\|^2$$

  $$\text{s.t.} \ \ y_i \left(w^T x_i + b\right) \geq 1, \ \ \forall i = 1, ..., N.$$

  La desigualdad representa la restricción de que los puntos de datos deben estar clasificados correctamente.

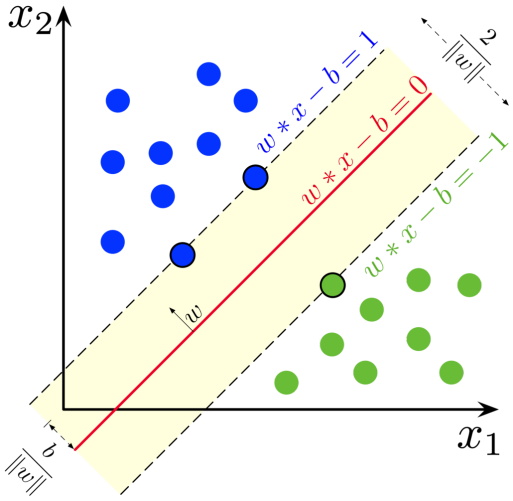


Figura 22: Hard-Margin SVM.

  Sin embargo, el hard-margin SVM tiene una limitación: solo funciona cuando los datos son linealmente separables. En la práctica, muchos conjuntos de datos son ruidosos o contienen errores, lo que hace que la separación perfecta no sea posible.

- **Con margen suave** (*Soft-Margin SVM*)

Este modelo es una extensión del hard-margin SVM que permite clasificar conjuntos de datos que **no son completamente linealmente separables**. En lugar de buscar un hiperplano perfecto, se permite un cierto grado de error en la clasificación de los puntos de datos.

El soft-margin SVM introduce **variables de holgura** o *slack variables* (ξ_i) que cuantifican la cantidad de error o violación de las restricciones de clasificación para cada punto de datos. Estas variables de holgura permiten que algunos puntos de datos estén dentro o más allá del margen o incluso en el lado incorrecto del hiperplano.

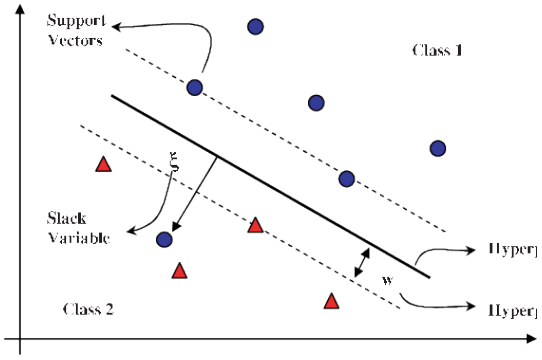


Figura 23: Soft-Margin SVM.

El objetivo del soft-margin SVM es encontrar un hiperplano que minimice el error y maximice el margen al mismo tiempo. Esto se puede lograr formulando el problema de optimización de la siguiente manera

$$\min \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{s.t.} \ \ y_i(w^T x_i + b) \geq 1 - \xi_i, \ \forall i = 1, ..., N,$$

$$\xi_i \geq 0, \ \forall i = 1, ..., N.$$

Aquí, el parámetro C controla el equilibrio entre el margen y la penalización por error. Un valor mayor de C penalizará más los errores de clasificación (para valores grandes de C, se comportará de manera similar a la hard-margin SVM), lo que resultará en un margen más estrecho, mientras que un valor menor de C permitirá un margen más amplio con más errores de clasificación.

En algunos casos, los datos pueden no ser linealmente separables en el espacio de características original. Aquí es donde entra en juego el **Kernel SVM**.

A.6.1. Kernel SVM

El **Kernel Support Vector Machine** (*Máquinas de Vectores de Soporte Kernelizadas* es castellano), más conocido como *Kernel SVM*, es una extensión del Support Vector Machine que permite la **clasificación no lineal** de conjuntos de datos. En lugar de tratar de encontrar un hiperplano lineal en el espacio original de características, el Kernel SVM

utiliza una **función de kernel** para mapear los datos a un espacio de mayor dimensión, donde puede ser posible una separación lineal.

El objetivo del Kernel SVM es encontrar un hiperplano óptimo en el espacio de características transformado que pueda separar eficientemente los datos de diferentes clases.

Matemáticamente, consideremos un conjunto de datos de entrenamiento (x_i, y_i), donde x_i es el vector de características y y_i es la etiqueta de clase correspondiente. El objetivo es encontrar una función de decisión $f(x)$ en el espacio transformado que nos permita clasificar nuevos puntos de datos.

La función de decisión en el espacio transformado se define como

$$f(x) = w^T \Phi(x) + b,$$

donde $\Phi(x)$ es una función de mapeo no lineal que transforma los datos al espacio de características de mayor dimensión.

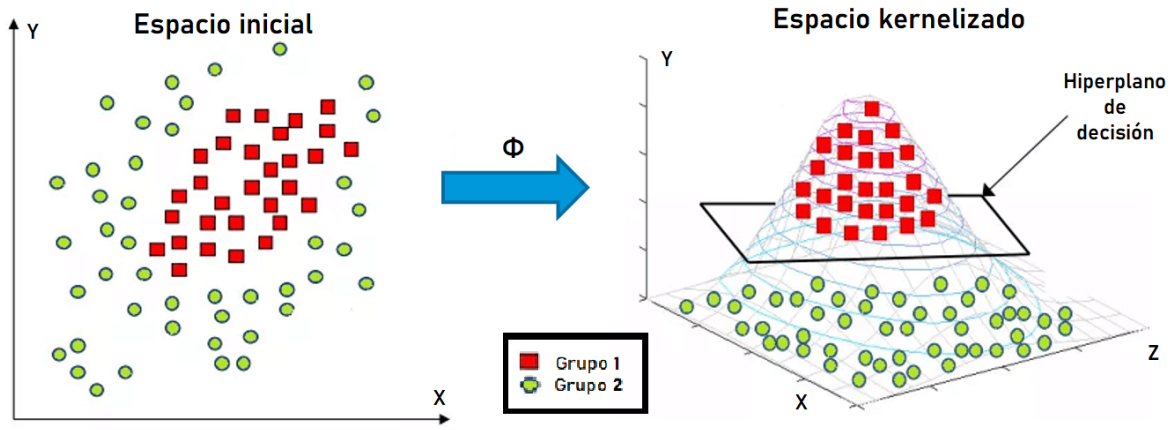


Figura 24: Kernel SVM.

El truco del kernel permite calcular el producto interno en el espacio de características de mayor dimensión sin tener que calcular explícitamente las coordenadas de los datos en ese espacio. Dado un kernel K, el producto interno se puede expresar como

$$\phi(x_i)^T \phi(x_j) = K(x_i, x_j).$$

Algunos ejemplos comunes de funciones de kernel son:

- **Kernel Lineal**: Este es el kernel más simple y se utiliza cuando los datos son linealmente separables en el espacio original de características. El producto interno entre

los vectores de características se utiliza directamente como medida de similitud. La función de kernel lineal se define como

$$K(x, x') = x^T x'.$$

Tal y como se puede observar, este es el SVM explicado anteriormente.

- **Kernel Polinomial**: Este kernel se utiliza para mapear los datos a un espacio de características de mayor dimensión mediante una función polinomial. Permite capturar relaciones no lineales en los datos. La función de kernel polinomial se define como

$$K(x, x') = (x^T x' + c)^d,$$

donde d es el grado del polinomio y c es un término constante.

- **Kernel RBF** (*Radial Basis Function*): El kernel RBF, también conocido como *kernel gaussiano*, es uno de los kernels más utilizados debido a su capacidad para manejar relaciones no lineales en los datos. La función de kernel RBF se define como

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right),$$

donde x y x' son dos puntos de datos, $\|x - x'\|$ representa la distancia euclidiana entre ellos y σ es un parámetro que controla la amplitud de la función RBF.

Este kernel asigna un valor de similitud o cercanía entre dos puntos de datos. Cuanto más cercanos estén los puntos, mayor será el valor de similitud, y a medida que aumente la distancia, el valor de similitud disminuirá. Esto permite capturar relaciones complejas en los datos y lograr una clasificación más precisa.

El parámetro σ en el kernel RBF controla la influencia de los puntos de datos vecinos en la clasificación. Un valor pequeño de σ dará como resultado una función de kernel RBF más "dura", donde solo los puntos de datos muy cercanos tendrán un impacto significativo en la clasificación. Por otro lado, un valor grande de σ producirá una función más "suave", donde incluso los puntos de datos más alejados pueden tener una influencia considerable en la clasificación.

Estos son solo algunos de los kernels más comunes utilizados en el Kernel SVM. La elección del kernel depende de la naturaleza de los datos y de la complejidad de las relaciones no lineales que se deseen capturar. Cada uno tiene sus propias características y propiedades matemáticas que los hacen adecuados para diferentes tipos de problemas de

clasificación. Si se desea profundizar más acerca de estos o de otros métodos de kernels se recomienda la lectura [17].

En resumen, el SVM y el Kernel SVM son algoritmos que buscan encontrar el hiperplano óptimo o la función de decisión que separa los datos en diferentes clases. La diferencia es que el Kernel SVM utiliza una función de kernel para transformar los datos a un espacio de características de mayor dimensión, lo que permite encontrar hiperplanos no lineales en ese espacio.

A.7. Redes neuronales artificiales

Una **red neuronal biológica** está formada por neuronas, interconectadas unas con otras de forma ponderada, de manera que cuando son estimuladas eléctricamente transmiten una señal a través del axón. Esta señal no es transferida a la siguiente neurona directamente, sino que sufre una transformación en la sinapsis. La neurona receptora convierte las señales de salida de las neuronas precedentes en una única señal de entrada. Finalmente, dependiendo de cómo la neurona es estimulada por la entrada acumulada, esta produce una respuesta emitiendo o no un pulso.

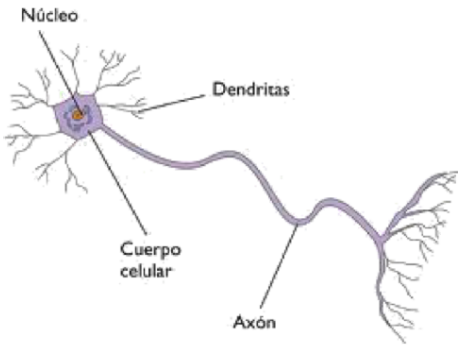


Figura 25: Neurona natural.

Por otro lado, una **red neuronal artificial** o ANN (por sus siglas en inglés *Artificial Neural Network*) es un modelo matemático y computacional que se inspira en el funcionamiento de las redes neuronales biológicas presentes en el cerebro humano.

Tal y como se observa en la Figura 26, las neuronas artificiales no son más que un elemento simple de un esquema mayor que son las redes neuronales. Estas redes consisten en unidades de proceso muy simples, distribuidas paralelamente y conectadas entre sí a través de uniones que están moduladas por pesos.

Matemáticamente, una red neuronal artificial es una terna (N, V, W), donde

- N es el conjunto formado por las neuronas artificiales, también llamados **nodos**.
- V es el conjunto de conexiones dirigidas (i, j) que une la neurona i con la j, también llamadas **aristas**.

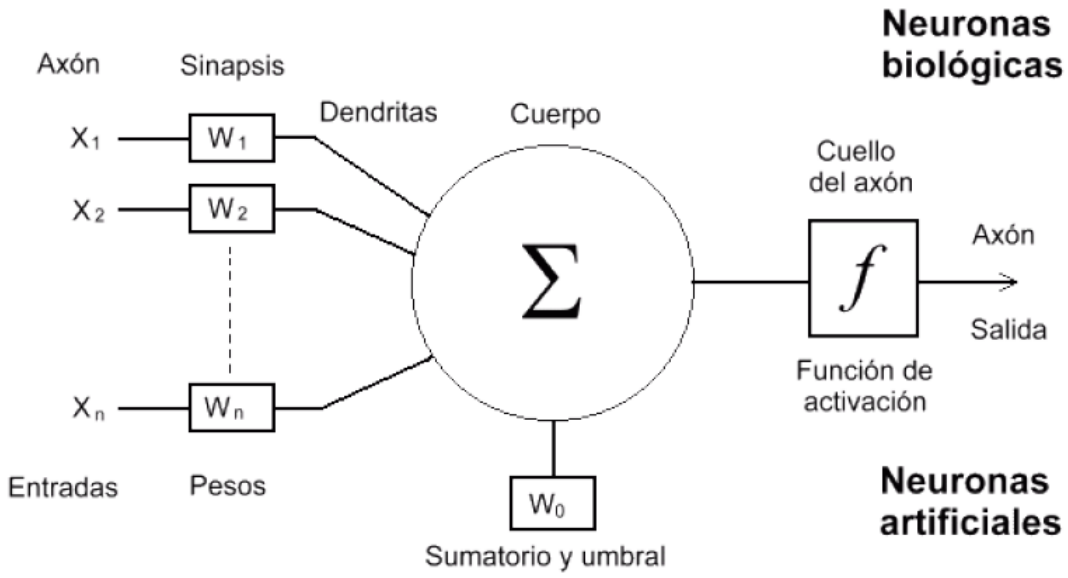


Figura 26: De la neurona natural a la artificial.

- W es la matriz que reune los pesos entre las neuronas, también llamada **matriz de pesos**.

Las ANN se caracterizan por:

- **Arquitectura**: estructura o patrón de las conexiones entre las neuronas.
- **Dinámica de la computación**: dice el valor que toman las neuronas que componen la red en base a las funciones de activación.
- **Algoritmo de entrenamiento o aprendizajes**: procedimiento para determinar los pesos entre las diferentes conexiones.

A.7.1. Arquitectura de las redes neuronales artificiales

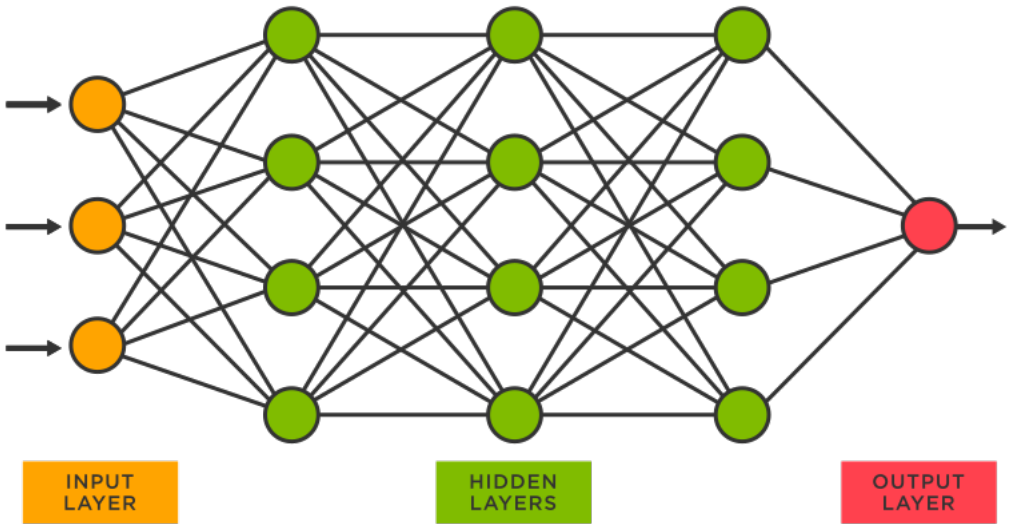
Para formar la arquitectura de una red neuronal, primero se tienen que formar las **capas** que la componen. Estas son componentes estructurales que organizan y procesan la información a medida que se propaga a través de la red. Cada capa está compuesta por un conjunto de neuronas o unidades de procesamiento que realizan operaciones en paralelo.

Existen tres tipos principales de capas:

- **Capa de entrada** (*input layer*): Esta es la capa inicial de la red y es responsable de recibir los datos de entrada. Cada neurona en esta capa representa una característica o atributo específico de los datos de entrada.

- **Capas ocultas** (*hidden layers*): Estas capas se encuentran entre la capa de entrada y la capa de salida. Son llamadas "ocultas" porque sus salidas no son directamente observables y no están relacionadas con los datos de entrada o de salida de forma inmediata. Estas capas son responsables de realizar transformaciones no lineales de las entradas y extraer características o representaciones intermedias de los datos. Cuantas más capas ocultas tenga la red, mayor será su capacidad para capturar relaciones complejas en los datos, pero mayor coste computacional tendrá el proceso de entrenamiento. Se llama **red neuronal monocapa** a una red neuronal artificial que únicamente tiene una capa oculta, y **multicapa** a una red neuronal artificial con más de una capa oculta.

- **Capa de salida** (*output layer*): Esta es la capa final de la red y produce las salidas finales de la red neuronal. Las neuronas en esta capa representan las clasificaciones, predicciones o valores continuos que se desean obtener como resultado de la red.



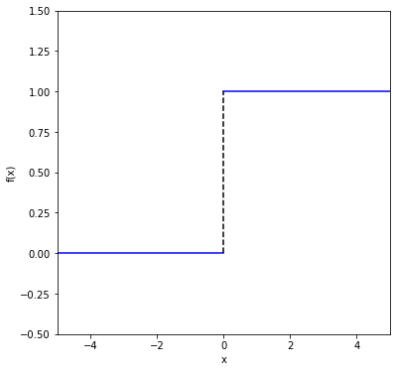
Una de las mayores complicaciones de las redes neuronales es elegir la mejor arquitectura para un problema dado, ya que esta depende del problema que se esté abordando y de la complejidad de los datos.

A.7.2. Funciones de activación

Otra componente fundamental en las redes neuronales son las **funciones de activación**, puesto que introducen no linealidades en el proceso de cálculo de cada neurona. Estas funciones determinan la salida de una neurona en función de la suma ponderada de las entradas recibidas.

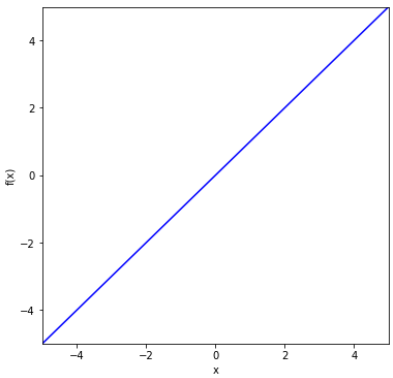
A continuación, se explican algunas de las funciones de activación más comunes utilizadas en las redes neuronales:

- **Función de paso de Heaviside** (*Función Escalón*): La función de paso de Heaviside se utiliza en situaciones donde se desea una respuesta binaria, como en la clasificación de problemas de dos clases. Esta función se define como



$$H(x) = \begin{cases} 0 & si \quad x < 0 \\ 1 & si \quad x \geq 0. \end{cases}$$

  Hoy en día no se utiliza mucho debido a su naturaleza discontinua, ya que no es diferenciable y no se puede utilizar en métodos de aprendizaje basados en el gradiente.

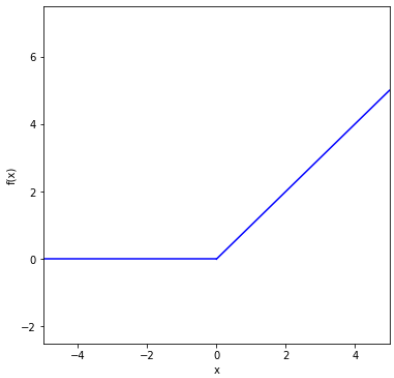


- **Función lineal**: La función lineal es una función de activación simple que produce una salida proporcional a la entrada. La fórmula de la función lineal es

$$f_L(x) = cx,$$

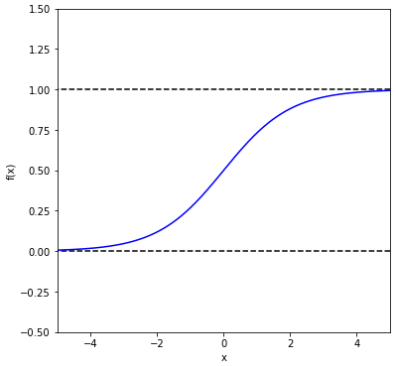
  donde c es una constante. A diferencia de las funciones no lineales, la función lineal no introduce no linealidad en la red neuronal y, por lo tanto, se utiliza principalmente en casos donde se requiere una salida lineal, como en regresión lineal.

- **Función ReLU** (*Rectified Linear Unit*): La función ReLU es una función de activación no lineal que retorna el valor de entrada si es positivo, y cero en caso contrario. Esta función viene dada por



$$f_R(x) = \max(0, x).$$

La función ReLU es una de las funciones más utilizadas en redes neuronales hoy en día debido a su simplicidad y capacidad para mitigar el problema del desvanecimiento del gradiente. Al eliminar los valores negativos, esta función permite que las neuronas sean más activas y promueve una mayor capacidad de aprendizaje.

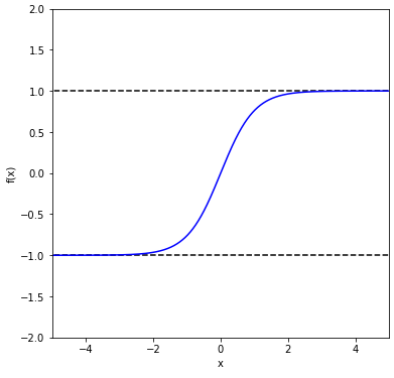


- **Función Sigmoide** (*Logística*): La función sigmoide es una función continua y diferenciable que mapea cualquier valor real a un rango entre 0 y 1. La fórmula de la función sigmoide es

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

Esta función se utiliza comúnmente en capas ocultas de redes neuronales, ya que es diferenciable y su salida se encuentra en el rango acotado, lo que facilita el entrenamiento de la red mediante algoritmos de optimización como el descenso de gradiente (*SGD*) o la estimación de momento adaptativo (*ADAM*). Cabe destacar que en problemas de clasificación binaria se suele utilizar en la capa de salida, pues al devolver un valor entre 0 y 1 se puede interpretar como la probabilidad.

- **Función Tangente Hiperbólica** (*Tanh*): La función tangente hiperbólica es similar a la función sigmoide, pero su rango está entre -1 y 1. La fórmula de la función tangente hiperbólica es



$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

Al igual que la función sigmoide, esta función es diferenciable y se suele utilizar en capas ocultas de las redes neuronales.

- **Función Softmax**: La función softmax se utiliza comúnmente en la capa de salida de una red neuronal cuando se realiza clasificación multiclase (para clasificación binaria se utiliza más la sigmoide, tal y como se ha explicado anteriormente). Sea x un vector K-dimensional, entonces esta función viene dada por

$$\sigma(x)_j = \frac{e_j^x}{\sum_{k=1}^{K} e^{x_k}} \quad \text{para } j = 1, \ldots, K.$$

  Es decir, la función softmax toma un vector de valores reales y produce un vector de probabilidades normalizadas en el rango de 0 a 1, donde la suma de todas las probabilidades es igual a 1. Por esto interpretar la salida de la red como una distribución de probabilidad sobre las diferentes clases.

Estas son solo algunas de las tantas las funciones de activación que se utilizan en las redes neuronales. Existen otras variantes y funciones especializadas que se utilizan en casos específicos. La elección de la función de activación adecuada depende del problema en cuestión, la naturaleza de los datos y las características deseadas en la salida de la red. Dicho esto, en el artículo [14] se proporciona una explicación sobre qué funciones de activación funcionan mejor en los diferentes tipos de datos.

A.7.3. Algoritmos de entrenamiento

De igual forma que el machine learning clásico, las redes neuronales pueden ser supervisadas y no supervisadas. En este trabajo únicamente se hablará de las **supervisadas**. En este tipo de aprendizaje, la red neuronal se entrena utilizando un conjunto de datos de entrenamiento etiquetados. Cada ejemplo de entrenamiento consta de una entrada y una salida deseada correspondiente. El objetivo de la red es aprender a mapear las entradas a las salidas correctas.

Como las entradas a las neuronas de una capa (oculta o de salida) son las salidas de las neuronas de la capa precedente (entrada u oculta respectivamente), la expresión de la salida vendrá dada por

$$y_i = f_1\Bigg(\underbrace{\sum_{j=1}^{H} w_{ij} s_j f(1)}_{h_i}\Bigg) = f_1\Bigg(\sum_{j=1}^{H} w_{ij} f_2\Bigg(\underbrace{\sum_{r=1}^{L} t_{jr} x_r}_{u_j}\Bigg)\Bigg), \quad \text{para } i = 0, \ldots, M, \tag{8}$$

donde

- w_{ij} es el peso sináptico de la conexión entre la neurona de salida i y la de proceso j de la capa oculta constituida por H neuronas.

- f_1 es la función de transferencia de las neuronas de la capa de salida que está formada por M neuronas.
- t_{jr} es el peso sináptico que conecta la neurona j de la capa oculta con la r de la capa de entrada constituida por L neuronas.
- f_2 es la función de transferencia de las neuronas de la capa oculta.

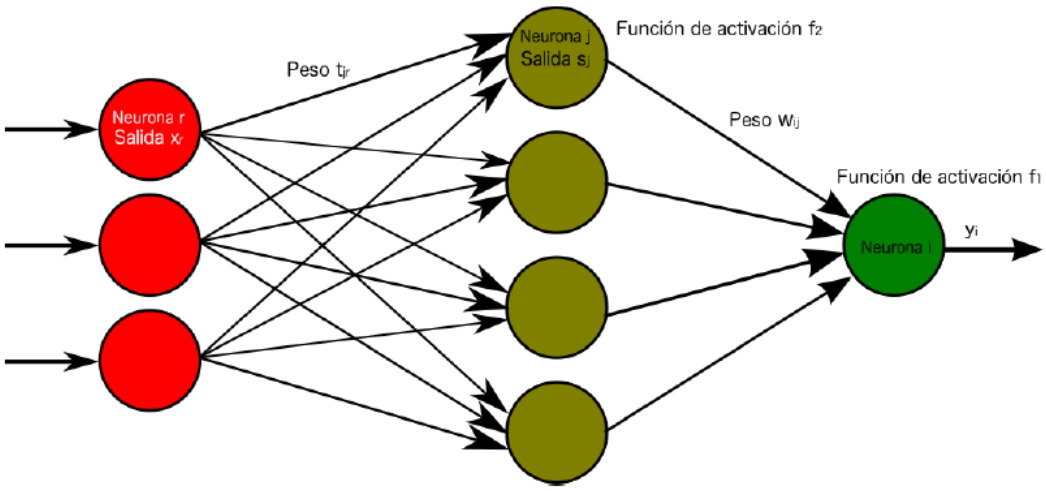
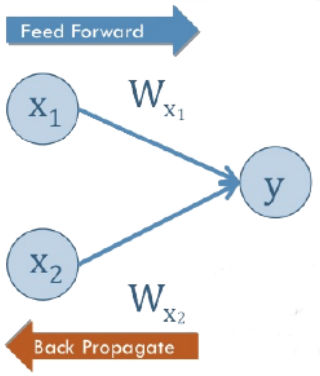


Figura 27: Ejemplo de ANN monocapa.

En este contexto, la primera pregunta que nos surge es, ¿cómo establecemos los valores de los pesos a partir de los patrones de entrenamiento en un aprendizaje supervisado?

El algoritmo más simple y comúnmente utilizado en el entrenamiento supervisado de redes neuronales es la **retropropagación**. La dificultad está en las capas ocultas, ya que no sabemos cuáles serían las salidas deseadas para esas capas.

La solución se basa en dos operaciones principales:



- **Feedforward**: consiste en presentar un patrón a las unidades de entrada y pasar las señales a través de la red para producir salidas.
- **Backpropagation**: utiliza el método de optimización elegido (hiperparámetro) para realizar un ajuste de los pesos, comenzando por la capa de salida, según el error cometido, propagando el error a las capas anteriores, de atrás hacia delante, hasta llegar a la capa de las neuronas de entrada.

En cuanto a los **métodos de optimización**, es importante saber cuál elegir dado el modelo. Los más comunes son el **método del descenso del gradiente** (*SGD*) (ver [1]) o la **estimación del momento adaptativo** (*ADAM*, por sus siglas en inglés **adaptive moment Estimation**) (ver [16]). Una diferencia entre estos es que el ADAM tiende a converger más rápido, mientras que SGD a menudo converge a soluciones más óptimas. Por tanto, para un tamaño de datos muy grande o un número de parámetros muy elevado conviene utilizar el método ADAM, mientras que para un tamaño de datos o parámetros pequeño conviene utilizar el SGD.

El algoritmo de retropropagación del error con el método del descenso del gradiente funciona de la siguiente manera:

1. **Inicialización de los pesos**: Se asignan valores iniciales aleatorios a los pesos de las conexiones entre las neuronas de la red neuronal.

2. **Propagación hacia adelante** (*Feedforward*): Se realiza un pase hacia adelante a través de la red neuronal. Los datos de entrada se propagan capa por capa, desde la capa de entrada hasta la capa de salida, calculando las salidas de cada neurona. Esto implica realizar una combinación lineal de las entradas ponderadas por los pesos y aplicar una función de activación no lineal para obtener las salidas de cada neurona.

3. **Cálculo del error**: Se calcula la diferencia entre las salidas obtenidas por la red neuronal y las salidas deseadas (etiquetas) utilizando una función de pérdida, que generalmente es el **error cuadrático medio** (mse). Esta función mide qué tan cerca o lejos están las salidas del resultado deseado y cuantifica el error de la red.

4. **Retropropagación del error** (*Backpropagation*): A partir de la capa de salida, se calcula el gradiente de la función de pérdida con respecto a los pesos de las conexiones y el sesgo de cada neurona. El gradiente representa la dirección y la magnitud del cambio necesario para reducir el error. Este cálculo se realiza utilizando la regla de la cadena y propagando el error hacia atrás a través de la red. Por tanto, aplicando el método del gradiente al ejemplo de la Figura 27 con ecuación (8) (ver [7]), los nuevos pesos entre la **capa oculta y la de salida** en la iteración k serán

$$w_{ij}(k+1) = w_{ij}(k) + \eta\delta_i^{(2)}(k)s_j(k) \quad \text{con} \quad \delta_i^{(2)}(k) = (z_i(k) - y_i(k))f_1'(h_i),$$

y los nuevos pesos entre la **capa de entrada y la oculta** en la iteración k serán

$$t_{jr}(k+1) = t_{jr}(k) + \eta\delta_j^{(1)}(k)x_r(k) \quad \text{con} \quad \delta_j^{(1)}(k) = f_2'(u_j)\sum_{i=1}^{M} w_{ij}(k)\delta_i^{(2)}(k).$$

5. **Actualización de los pesos**: Utilizando el gradiente calculado en el paso anterior, se actualizan los pesos y sesgos de las conexiones en la red. Esto se realiza utilizando el método del descenso del gradiente (u otro método de optimización), que ajusta los pesos en la dirección opuesta al gradiente para minimizar la función de pérdida.

6. **Repetición del proceso**: Los pasos 2 a 5 se repiten para cada ejemplo de entrenamiento en el conjunto de datos. Este proceso iterativo permite que la red neuronal aprenda gradualmente a través de múltiples iteraciones, ajustando los pesos y mejorando su capacidad para hacer predicciones más precisas.

7. **Terminación del entrenamiento**: El entrenamiento se detiene cuando se alcanza un criterio de terminación predefinido, como un número máximo de iteraciones o cuando el error de validación sea menor que cierto valor.

Para una ilustración con figuras más clara y detallada sobre un ejemplo de aplicación de este algoritmo en una red neuronal multicapa se recomienda entrar al siguiente enlace.

En resumen, el algoritmo de retropropagación del error consiste en propagar el error desde la capa de salida hacia la capa de entrada de una red neuronal, calculando los gradientes de los pesos y sesgos de cada neurona. Estos gradientes se utilizan para ajustar los pesos de la red mediante un algoritmo de optimización, con el objetivo de minimizar la función de pérdida y mejorar el rendimiento de la red en la tarea deseada.

Las ANN se utilizan en diferentes tipos, que, básicamente, se pueden agrupar en:

- **Predicción**: Se trata de determinar series temporales. Tenemos una sucesión de datos en tiempos $t \in \{1, 2, \ldots, N\}$ y queremos conocer el dato en un tiempo posterior $t = N + 1$, entonces la función de predicción viene dada por

$$\begin{aligned} f: \ \mathbb{R}^n &\longrightarrow \mathbb{R}^n \\ x &\longrightarrow x(t+1). \end{aligned}$$

- **Agrupamiento supervisado** (**clasificación**): Dado un conjunto de datos caracterizados por una serie de variables, que están categorizados en un conjunto de clases mediante ciertas etiquetas denotadas por $\{0, 1, \ldots, n\} \in \mathbb{B}$, queremos disponer de una función que, ante un dato nuevo, infiera a qué clase pertenece.

$$\begin{aligned} f: \ \mathbb{R}^n &\longrightarrow \mathbb{B} \\ x &\longrightarrow \{0, 1, \ldots, n\}. \end{aligned}$$

- **Agrupamiento no supervisado** (competitivo o autoorganizado): En este caso, a diferencia del anterior, desconocemos si los datos dados por esas variables se agrupan en diferentes categorías. Queremos una función que, ante la masa informe de datos, nos diga cuántas posibles categorías se podrían establecer a tener el valor de las variables que describen dichos datos.

$$\begin{aligned} f:\ \mathbb{R}^n &\longrightarrow (\mathcal{B}, \tau) \\ x &\longrightarrow \textit{etiqueta}. \end{aligned}$$

  Un algoritmo de aprendizaje no supervisado ampliamente utilizado en ANN es el algoritmo de **autoencoders**. En este trabajo no nos centraremos en este tipo de aprendizaje de redes, pero si el lector desea obtener una explicación de este algoritmo, recomendamos la lectura [9].

- **Aproximación** (**regresión**): Tenemos una serie de variables independientes y dependientes y queremos encontrar una aproximación a la función que describe la relación entre dichos grupos y variables.

$$\begin{aligned} f:\ \mathbb{R}^n &\longrightarrow \mathbb{R}^n \\ x &\longrightarrow y(x). \end{aligned}$$

En este proyecto se ha utilizado tanto la clasificación como la predicción.

A.8. LSTM

Las **memorias de corto-largo plazo**, comúnmente denominadas **LSTM** (*Long Short-Term Memory*) son un tipo especializado de red neuronal recurrente que se desarrolló por Hochreiter y Schmidhuber para superar el problema del decaimiento del gradiente en las RNN tradicionales y para capturar relaciones a largo plazo en datos secuenciales. Estas redes han demostrado ser muy efectivas en tareas que involucran secuencias de datos, como el procesamiento del lenguaje natural, la traducción automática y el reconocimiento de voz.

La arquitectura de una LSTM se basa en unidades de memoria, también conocidas como células LSTM, que contienen una estructura única que les permite mantener y actualizar información a lo largo del tiempo. Estas unidades de memoria están diseñadas para tener una puerta de entrada, una puerta de olvido y una puerta de salida, que controlan el flujo de información dentro de la célula LSTM.

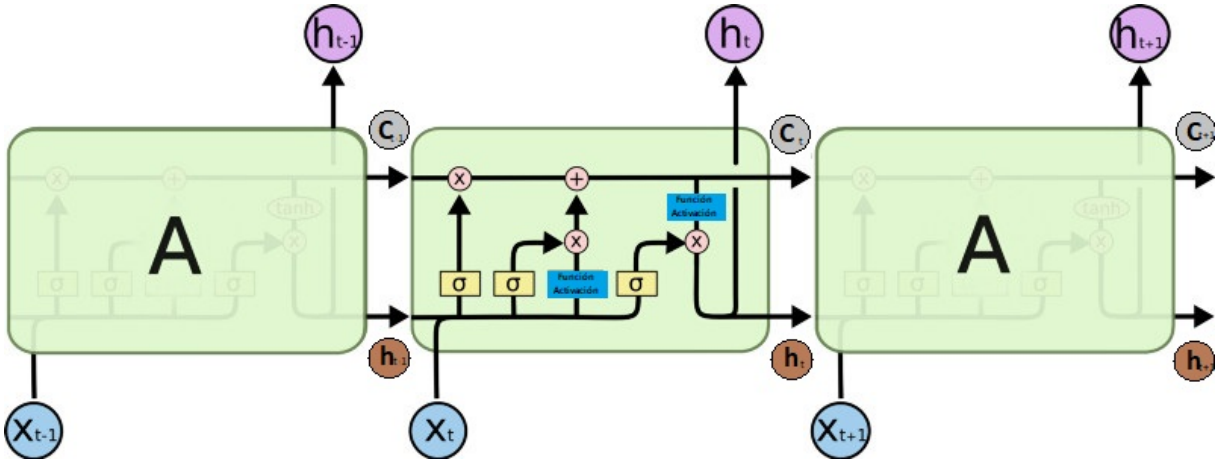


Figura 28: Arquitectura de una red LSTM.

El principio detrás del modelo LSTM es la utilización de células de memoria que permiten mantener un estado c separado del flujo normal de la red recurrente. De hecho, tal y como se observa en la Figura 28, el estado c tiene una conexión directa consigo mismo: el estado c_t es simplemente la suma del estado anterior c_{t-1}. Dado que la función de activación para actualizar el estado es la identidad, esto implica que la derivada de la función de activación utilizada para actualizar el estado es igual a 1, lo que evita que el gradiente desaparezca o se vuelva excesivamente grande durante la retropropagación. En lugar de ello, el gradiente se mantiene constante a lo largo del tiempo.

Para el procesamiento de datos, las LSTM implementan capas llamadas compuertas (o simplemente puertas) que aplican una función de activación sigmoidea a la concatenación de la entrada x_t y la salida h_{t-1}. Esto genera un vector de salida con elementos en el rango de $0 - 1$. Estas compuertas controlan la cantidad de información que se actualiza y se propaga en la red, permitiendo que las LSTM capturen y utilicen de manera selectiva la información relevante en cada paso de tiempo.

El funcionamiento de una LSTM se puede dividir en tres pasos principales:

1. **Olvido**: En esta etapa, la puerta de olvido determina cuánta información anterior debe ser descartada de la unidad de memoria. La puerta de olvido es controlada por una función de activación sigmoide, que va de 0 a 1. Un valor cercano a 0 indica que se debe descartar completamente la información anterior, mientras que un valor cercano a 1 indica que toda la información debe mantenerse.

2. **Almacenamiento**: En esta etapa, la puerta de entrada decide cuánta información nueva se debe agregar a la unidad de memoria. Primero, una función de activación

sigmoide determina qué valores se deben actualizar y luego se calcula una candidata de actualización. Luego, la puerta de entrada controla cuánta de esta información se debe almacenar en la unidad de memoria. Los valores antiguos se mezclan con la nueva información para actualizar el estado de la unidad de memoria.

3. **Salida**: En esta etapa, la puerta de salida decide cuánta información de la unidad de memoria se debe utilizar como salida. La salida se basa en el estado actual de la unidad de memoria y se filtra a través de una función de activación sigmoide.

En resumen, las redes neuronales LSTM utilizan unidades de memoria con puertas para mantener y actualizar información a lo largo del tiempo en tareas de procesamiento de datos secuenciales. Esto les permite capturar dependencias a largo plazo y ha demostrado ser muy efectivo en aplicaciones como el procesamiento del lenguaje natural y el reconocimiento de voz.

B. Librerias utilizadas

NumPy

La librería *NumPy* es una biblioteca fundamental en *Python* para el **procesamiento numérico y científico de datos**. Su nombre proviene de "Numerical Python". NumPy proporciona estructuras de datos eficientes y de alto rendimiento, como matrices multidimensionales (llamadas arrays) y funciones matemáticas para realizar operaciones numéricas rápidas y eficientes.

Por estas razones, en este trabajo se ha optado por la utilización de NumPy para el procesamiento numérico y científico de datos. Para más información visitar su Web oficial.

Matplotlib

La librería *Matplotlib* es una **biblioteca de visualización de datos** en *Python* ampliamente utilizada y altamente versátil. Proporciona una amplia gama de herramientas y funcionalidades para crear gráficos estáticos, interactivos y personalizados. Su versatilidad, su integración con *NumPy* y su capacidad de personalización hacen de Matplotlib una opción poderosa y flexible para la visualización de datos en entornos científicos, de análisis de datos y de desarrollo de aplicaciones.

Por estas razones, en este trabajo se ha optado por la utilización de Matplotlib para la visualización de datos. Para más información visitar su Web oficial.

Pandas

La librería *Pandas* es una de las bibliotecas más populares y ampliamente utilizadas en el ecosistema de *Python* para el **análisis y la manipulación de datos**. Proporciona estructuras de datos flexibles y eficientes para trabajar con datos numéricos y tabulares, así como herramientas para realizar operaciones de limpieza, transformación y análisis de datos.

Las características clave de Pandas incluyen estructuras de datos, manipulación de datos, tratamiento de datos faltantes, operaciones estadísticas y de análisis, entre otras.

Hemos optado por esta librería, ya que para trabajar con archivos "*csv*" esta funciona

muy bien. Para más informaciós visitar su Web oficial.

scikit-learn

La librería *scikit-learn* o *sklearn* es una **biblioteca de aprendizaje automático** de código abierto para el lenguaje de programación *Python*. Proporciona una amplia gama de algoritmos y herramientas para tareas comunes de minería de datos y análisis predictivo. "scikit-learn" se destaca por su facilidad de uso y su enfoque en la eficiencia y la modularidad.

Esta librería ofrece una variedad de funcionalidades, incluyendo:

- **Implementación de algoritmos de aprendizaje supervisado**: scikit-learn incluye implementaciones de diversos algoritmos supervisados, como regresión lineal, regresión logística, árboles de decisión, SVM, entre otros.
- **Algoritmos de aprendizaje no supervisado**: La librería también proporciona algoritmos de aprendizaje no supervisado, como agrupamiento, reducción de dimensionalidad y detección de anomalías.
- **Preprocesamiento de datos**: scikit-learn ofrece herramientas para el preprocesamiento y la transformación de los datos antes de aplicar los algoritmos de aprendizaje. Esto incluye la normalización de datos, la selección de características relevantes, la codificación de variables categóricas, entre otros.
- **Evaluación y selección de modelos**: La librería proporciona métricas y herramientas para evaluar el rendimiento de los modelos de aprendizaje.
- **Validación cruzada**: scikit-learn incluye funcionalidades para realizar validación cruzada. Esto ayuda a estimar el rendimiento del modelo de manera más precisa y mitigar problemas como el sobreajuste (overfitting).

Además de estas funcionalidades principales, scikit-learn también ofrece herramientas para el ajuste de hiperparámetros, el manejo de conjuntos de datos, la generación de datos sintéticos, la visualización y más.

Es por todas estas ventajas por lo que se ha optado esta librería para la parte del procesamiento de datos y los algoritmos de ML tradicional. Para más informaciós visitar su Web oficial.

Tensorflow

La librería *TensorFlow* es una **biblioteca de código abierto para el aprendizaje automático y la inteligencia artificial** desarrollada por *Google*. Se utiliza ampliamente en la comunidad de investigación y desarrollo de aprendizaje automático debido a su potencia, flexibilidad y escalabilidad. Esta librería se centra en la construcción y entrenamiento de modelos de aprendizaje profundo, aunque también admite otros tipos de algoritmos de aprendizaje automático.

Estas son algunas de las características y funcionalidades clave de TensorFlow:

- **Grafos computacionales**: TensorFlow utiliza grafos computacionales para representar cálculos. Esto permite una ejecución eficiente de cálculos en CPUs, GPUs y otros dispositivos de cómputo acelerado.
- **Abstracción de tensores**: Esta librería se basa en el concepto de tensores, que son arreglos multidimensionales de datos. Los tensores son la base fundamental de los cálculos en TensorFlow y permiten representar y manipular datos en múltiples dimensiones de manera eficiente.
- **Construcción de modelos**: TensorFlow proporciona una API flexible y modular para la construcción de modelos de aprendizaje profundo. Los modelos se definen mediante la composición de capas, donde cada capa realiza operaciones específicas, como convoluciones, agrupamientos o capas completamente conectadas.
- **Entrenamiento y optimización**: Esta librería ofrece herramientas y algoritmos para el entrenamiento de modelos mediante la optimización de una función de pérdida. Proporciona optimizadores como el descenso de gradiente estocástico (SGD) y algoritmos más avanzados como el Adam Optimizer. Además, TensorFlow facilita el cálculo automático de gradientes, lo que simplifica la implementación de algoritmos de retropropagación como son las ANN.

En resumen, TensorFlow es una poderosa biblioteca de aprendizaje automático y aprendizaje profundo que ofrece un conjunto de herramientas flexibles y eficientes para la construcción, entrenamiento y despliegue de modelos. Es por su enfoque en grafos computacionales y su capacidad de optimización en algoritmos de inteligencia artificial que en este trabajo se ha optado por usar esta librería. Para más información visitar su Web oficial.

Keras

La librería *Keras* es una **biblioteca de aprendizaje profundo de alto nivel** escrita en *Python* que se utiliza ampliamente en el campo del aprendizaje automático y la inteligencia artificial. Estas son algunas de las características y funcionalidades clave de Keras:

- **Abstracción de redes neuronales**: Keras proporciona una interfaz de alto nivel para construir redes neuronales. Permite crear redes neuronales de forma rápida y sencilla mediante la composición de capas. Las capas pueden ser apiladas y conectadas de manera flexible para construir arquitecturas de red complejas.
- **Modularidad**: Keras ofrece una estructura modular en la que las capas y los modelos se pueden combinar y reutilizar fácilmente. Esto facilita la construcción de redes neuronales personalizadas y la experimentación con diferentes arquitecturas.
- **Soporte para múltiples backends**: Keras es compatible con varios backends, incluyendo *TensorFlow*. Esto significa que se puede elegir el backend de preferencia para ejecutar los cálculos y aprovechar las ventajas y características de cada uno.
- **Soporte para redes neuronales feedforward, recurrentes y otras**: Keras proporciona capas y funciones específicas para trabajar con diferentes tipos de redes neuronales.
- **Facilidad de extensión**: Keras permite extender su funcionalidad mediante la creación de capas y modelos personalizados. Esto brinda flexibilidad para adaptarse a necesidades específicas y experimentar con nuevas ideas en el campo del aprendizaje profundo.
- **Integración con otras herramientas de aprendizaje profundo**: Keras se integra de manera fluida con otras bibliotecas y herramientas populares de aprendizaje profundo, como TensorFlow. Esto permite combinar las capacidades de Keras con las funcionalidades más avanzadas y de bajo nivel de otras herramientas.

En general, Keras es una librería poderosa y amigable para la construcción y entrenamiento de redes neuronales. Es por su simplicidad y facilidad a la hora de crear y entrenar redes neuronales de manera eficiente que en este trabajo se ha optado por usar esta librería. Para más información visitar su Web oficial.