Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

Máster Universitario en Ingeniería Industrial (MUEI)

Trabajo de Fin de Máster

Clasificación de la relación social en parejas durante el acompañamiento

22 de septiembre de 2021

Autor: Óscar Castro Arcusa

Directora: Ely Repiso Polo

Ponente: Anaís Garrell Zulueta

Convocatoria: Septiembre 2021

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Para triunfar, el solo planteamiento es insuficiente. También se debe improvisar.

Isaac Asimov

Resumen

En este documento se recoge la explicación y diseño de diversos métodos que permiten clasificar la relación social existente entre dos personas que están acompañándose en cuatro posibles categorías (colegas, pareja, familia o amistad). Los métodos desarrollados se basan en técnicas de *clustering* y *deep learning*. En los métodos de *deep learning*, se implementan diferentes modelos que utilizan redes neuronales, tanto estándares como recurrentes, para llevar a cabo la clasificación.

Los resultados obtenidos a partir de los modelos diseñados se analizan y comparan con el objetivo de encontrar el modelo que presenta la mejor eficacia en el proceso de clasificación. También se comparan los resultados de los clasificadores con los resultados de un estudio similar.

Además, se revisa el estado del arte actual del mundo de la robótica social y otros campos relacionados y se realiza la planificación temporal de la tesis. Finalmente, se realiza un estudio económico del proyecto y se evalúa su posible impacto ambiental y social.

Todos los programas del proyecto se realizan en lenguaje Python y los diversos modelos de redes neuronales se programan utilizando TensorFlow y Keras. Además, se trabaja con una base de datos de personas acompañándose unas a otras clasificadas en cuatro posibles categorías (colegas, pareja, familia o amistad), extraída del estudio previo con el que se comparan los resultados obtenidos [1].

Índice

Índice de figuras

1. Introducción

En los últimos años, el mundo de la robótica está experimentando un crecimiento sin precedentes. Los avances tecnológicos en informática, *big data* o sistemas de percepción están permitiendo el desarrollo de robots capaces de encargarse de tareas que parecían solo posibles en la ciencia ficción unos años atrás.

En una sociedad donde cada vez más los humanos y los robots comparten el mismo espacio, la interacción entre ambos se presenta como uno de los grandes retos a los que se debe hacer frente. Por ello, se abre la posibilidad de que los robots se encarguen de tareas que previamente parecían destinadas a ser realizadas por humanos. En consecuencia, en un futuro no muy lejano se espera ver a robots y humanos trabajando juntos en tareas cotidianas mientras comparten áreas del espacio público.

Un robot móvil podría ser capaz de acompañar a un humano durante situaciones habituales, como hacer la compra [15], viajar a otros lugares [16] o visitar museos [17]. De la misma forma que los humanos se relacionan de manera diferente en función de la persona con la que estén interactuando, cada humano se comportará de manera distinta en función del robot con el que esté interactuando, y el tipo y tiempo de interacción.

Para conseguir que un robot que esté acompañando a una persona se adapte de la mejor manera posible, es necesario que este sea capaz de analizar, aprender y ejecutar las variables que implican los procesos de acompañamiento para dar una respuesta adecuada a cada persona y situación, para así obtener una interacción cómoda y segura para ambas partes.

1.1. Motivación

Hace unos años los robots parecían artefactos de un futuro lejano, pero hoy en día cuesta encontrar una planta productiva o incluso una casa sin uno de ellos. A medida que la tecnología avanza, las posibilidades que los robots ofrecen crecen de manera exponencial.

Uno de los pilares que la tecnología tiene que tener siempre presente es facilitar la vida de las personas. De esta forma, la interacción entre robots y personas debe realizarse de forma natural y segura. Para conseguir cumplir con esto, un robot tiene que ser capaz de identificar cómo se está comportando la persona a la cual está acompañando.

Gracias a la correcta identificación del comportamiento de la persona acompañada, el robot puede ser capaz de adaptar su movimiento para generar una mayor sensación de seguridad y confort a la persona y, por lo tanto, mejorar su experiencia durante el proceso.

En primer lugar, es necesaria la realización de un programa capaz de clasificar la relación entre dos personas a partir de las características observables durante un proceso de acompañamiento a lo largo del tiempo para, posteriormente, poder usar ese aprendizaje en los robots. Dicho aprendizaje les permitirá clasificar el comportamiento humano durante los primeros instantes de la interacción y poder así obtener un mejor y más personalizado acompañamiento de personas.

1.2. Objetivos

El objetivo principal del presente proyecto es clasificar la relación entre dos personas a lo largo de un proceso de acompañamiento. Este objetivo principal se puede desglosar en objetivos secundarios más concretos según las etapas del proyecto:

- Estudiar el estado del arte actual relacionado con la robótica social.
- Investigar y estudiar los posibles métodos de clasificación a utilizar.
- Implementar y modificar varios métodos de clasificación de relaciones sociales durante el acompañamiento entre personas (*clustering* y *deep learning*).
- Analizar y comparar los resultados obtenidos por los distintos métodos de clasificación.
- Realizar el estudio económico del proyecto.
- Evaluar el posible impacto ambiental y social.

2. Estado del arte

Con el paso de los años, el mundo de la robótica y el de los humanos están cada vez más conectados. Los avances científicos y tecnológicos están permitiendo desarrollar nuevos robots con funcionalidades que hace no mucho parecían difíciles de creer. Además, la sociedad actual está evolucionando y las nuevas generaciones están cada vez más concienciadas para incluir robots en su vida cotidiana. Por tanto, es necesario desarrollar robots capaces de colaborar y relacionarse con los seres humanos para realizar tareas diarias [18,19], como pueden ser acompañar [20,21], guiar [22], transportar [23] o limpiar [24].

Con el objetivo de que los robots móviles puedan ser introducidos de forma segura en ambientes poblados por humanos e incorporados en ellos de manera natural, estos deben ser diseñados teniendo en cuenta que deberán ser máquinas sociables [19, 25–27], una de las características básicas que define a los seres humanos y sus interacciones. Por tanto, bastantes estudios de las siguientes subsecciones del estado del arte tienen en cuenta la realización de las tareas de la forma más social posible [28–32].

Para conseguir que los robots sociales compartan espacio con los humanos en un futuro no muy lejano, estos deben desarrollar varias habilidades, como ser capaces de realizar una navegación con conciencia social. Algunas investigaciones en robótica han llegado a la conclusión de que parece que el comportamiento de las personas compartiendo espacios con robots no es tan diferente de la forma en la que se comportan con otras personas [33]. Por lo tanto, los robots sociales deberán ser capaces de seguir las convenciones sociales válidas para las sociedades humanas, y tendrán que ser diseñados con las habilidades necesarias para interactuar con individuos y grupos de personas.

En esta sección, se describe el estado del arte actual en diversos campos de la robótica o relacionados con ellos. En particular, se investiga en profundidad sobre los campos de la navegación social en la subsección 2.1; el comportamiento social entre personas, concretamente aplicado a la navegación con robots, en la subsección 2.2; el acompañamiento de personas llevado a cabo con robots sociales en la subsección 2.3; y la clasificación de relaciones sociales entre personas en la subsección 2.4.

2.1. Navegación social

Para los seres humanos, saber moverse por un entorno y alcanzar un destino específico ha sido vital para nuestra supervivencia y es indispensable para nuestro día a día. Cuando una persona se mueve por una zona, tanto conocida como desconocida, necesita conocer cierta información para ser capaz de navegar de forma eficiente por el espacio, como por ejemplo conocer su ubicación actual, la ubicación del destino objetivo y posibles caminos para ir a ese lugar. Del mismo modo, para un robot móvil la tarea de la navegación se plantea como una función básica y primordial. Por ese motivo, uno de los principales campos de estudio de la robótica siempre ha sido la navegación.

Chatila et al. [34] define la navegación como *"la tarea de alcanzar un objetivo distante arbitrario, sin que el camino haya sido predefinido o dado al robot"*. En [29] también define la navegación como el *"conjunto de acciones que se realizan para llegar a una meta asignada o destino desde un punto de origen*

determinado". Aplicando las definiciones anteriores al campo de la robótica, cualquier robot que tenga como objetivo desplazarse de una ubicación a otra precisa de un sistema de navegación que le permita moverse desde un punto inicial a un destino objetivo final.

En [35], los autores plantean que un robot móvil con plena capacidad de navegación debería ser capaz de pasar el siguiente test: *"Se coloca un robot en un entorno desconocido, amplio, complejo y dinámico. Una vez que el robot ha explorado el entorno, éste deber ser capaz de ir a cualquier lugar del entorno, tratando de minimizar una función de coste (por ejemplo: tiempo, energía, etc.)"*. Para cumplir este objetivo, en [36] se señala la necesidad que el robot pueda realizar diferentes acciones para responder a las preguntas que surgen del problema de la navegación: ¿Dónde estoy?, ¿A dónde voy? y ¿Cómo llego allí?

De un lado, la primera pregunta hace referencia al problema de localización y cómo puede desenvolverse el robot en el entorno en el que se encuentra en base a lo que puede ver y la información que previamente se le ha dado [37,38]. Por otro lado, la segunda y la tercera pregunta se centran en especificar un destino como objetivo y que el robot sea capaz de planificar una trayectoria que resulte en alcanzar dicho destino [39], haciendo frente a diversas tareas al mismo tiempo, como por ejemplo evitar colisiones con los elementos del entorno (obstáculos, peatones, etc.) [40], actualizar y predecir la información de su entorno [41], y calcular la trayectoria óptima para minimizar costes [42,43].

2.2. Comportamiento social entre personas aplicado a la navegación de robots

Como se introdujo anteriormente, uno de los aspectos fundamentales para que los robots interactúen con humanos de forma natural es que sean diseñados para ser máquinas sociables. En [44] se define la sociabilidad como *"todos los aspectos que hacen a los individuos interactuar entre ellos para satisfacer necesidades que no pueden lograr por sí solos. En contraste a la simple agrupación de individuos en unas condiciones ambientales favorables, la sociabilidad implica interacciones entre individuos"*.

Teniendo en cuenta la definición anterior, para que un robot sea social debe dejar de percibir a las personas de su alrededor tan solo como obstáculos dinámicos y empezar a considerarlas también como entidades sociales y elementos del entorno con los que poder interactuar. Esta consideración abre un nuevo campo de investigación en materia de navegación conocida como navegación con conciencia social.

En [31] se define la navegación con conciencia social como *"la estrategia exhibida por un robot social que identifica y sigue las convenciones sociales (en términos de gestión del espacio) con el fin de preservar una cómoda interacción con los humanos. El comportamiento resultante es predecible, adaptable y de fácil comprensión por humanos"*. Por lo tanto, un robot que lleve a cabo una navegación con conciencia social deberá tener como objetivo principal la interacción social con los humanos de su entorno, aparte de la navegación en sí misma.

Para profundizar en las interacciones sociales de los humanos, es necesario definir el concepto de convención social. En [45] se definen las convenciones sociales como *"las reglas y normas arbitrarias que gobiernan los innumerables comportamientos en los que todos nos involucramos todos los días sin pensar necesariamente en ellos"*. Estas normas permiten a las personas comportarse de forma aceptada por la inmensa mayoría de la sociedad y facilitan al resto de personas del

entorno comprender las intenciones de los individuos.

De esta forma, las convenciones sociales serán un factor especialmente importante a tener en cuenta durante la tarea de navegación del robot, teniendo que mostrar este un comportamiento seguro, comprensible y cómodo para los humanos de su entorno durante el desarrollo de la misma.

No obstante, las convenciones sociales dependen de diversos y variados factores que las hacen difíciles de generalizar para cualquier situación. En general, estas dependerán del entorno particular en que se lleve a cabo la navegación, teniendo en cuenta factores como el país o la cultura, pero también considerando otros factores tales como las características físicas del robot o las tareas que esté realizando.

Según [46], la navegación de un robot social debe sustentarse en tres pilares:

- Debe ser segura para asegurar que no hiere a los humanos.
- Debe ser fiable y efectiva, logrando completar las tareas de forma adecuada considerando las capacidades motrices del robot.
- Debe ser socialmente aceptable, para que tenga en cuenta el confort de los humanos, así como sus preferencias y necesidades, y que exprese la intención del robot claramente sin necesidad de comunicación extraordinaria.

Cuando se hace referencia a una navegación "segura", se hace referencia al significado más físico de la palabra, es decir, evitar colisiones y accidentes que puedan herir a los humanos o dañar al robot en sí mismo. Aun así, aunque los sistemas de navegación presenten métodos para prevenir colisiones, si un robot no es capaz de comunicar dicha seguridad mediante señales sociales que permitan a los humanos interpretarlas y sentirse seguros, la comodidad de estos se verá alterada y, posiblemente, disminuida [47].

Por lo tanto, tal y como se afirma en [48], la navegación con conciencia social tiene que ser adecuada para los humanos en entornos dinámicos, produciendo un comportamiento entendible y socialmente aceptable. El robot tiene que tener las habilidades necesarias para comportarse de manera que los humanos de su entorno no sientan miedo o se sientan incómodos por los movimientos del robot y se puedan identificar sus intenciones claramente. Además, los robots sociales deben diferenciar entre las personas de su alrededor y los otros obstáculos del entorno para poder comportarse de forma adecuada con ellas y poder, por ejemplo, mantener una velocidad y distancia adecuadas con las personas.

Dado que las personas y los robots navegarán compartiendo el mismo espacio físico, la capacidad que demuestren los robots para adaptar su navegación y comportamiento acorde a las convenciones sociales esperadas determinará el éxito o fracaso de sus aplicaciones.

En [31] se parte de la idea de que las personas mantendrán las mismas convenciones de la gestión del espacio social cuando interactúen con robots que cuando interactúan con humanos. A lo largo de los años, el comportamiento humano se ha estudiado desde varias perspectivas: psicología, sociología, antropología y neurociencia, con cada campo utilizando diferentes metodologías y criterios. Con el objetivo de diseñar una estrategia de navegación que permita a un robot realizar una navegación con conciencia social, es necesario intentar comprender cómo la

gestión del espacio que realizan los humanos al navegar afecta a su confort y cómo este podría verse afectado por la navegación de otros peatones o robots.

Una de las referencias más utilizadas en el campo de la navegación con conciencia social son las llamadas *proxemics*, o distancias sociales, propuestas por Hall en [28]. Hall observó la existencia de ciertas reglas no escritas que llevan a las personas a mantener cierta distancia con el resto de individuos y guían a estos a respetar esta distancia. De esta forma, acuñó el término *proxemics* para definir el estudio científico del espacio como medio de comunicación interpersonal, es decir, el estudio de distancias espaciales que los individuos mantienen en su entorno en varias situaciones sociales e interpersonales, dependiendo del entorno, factores culturales y otras características.

Concretamente, se define el espacio personal como *"la zona que rodea a un individuo, en donde no puede entrar otra persona sin autorización"* [49] y, en caso de invadir dicha zona, el confort del individuo se vería afectado. Esta zona puede ser variable de acuerdo a las circunstancias, a la persona o, en general, a las convenciones sociales.

Además, Hall propone una clasificación del espacio personal basada en cuatro zonas de distancia alrededor de una persona, representadas en la Figura 1. Al mismo tiempo, cada una de las zonas presenta una fase cercana y una fase lejana. Las características de las diferentes zonas se detallan a continuación:

a. **Zona íntima**: en la fase cercana predomina la posibilidad de una relación física estrecha (prácticas amorosas, deportes de contacto, confidencias). En la fase lejana hay una distancia de 15 a 45 cm en la que los cuerpos no entran fácilmente en contacto, pero las manos pueden alcanzar y asir las extremidades y hay deformación visual de los rasgos de las personas. Generalmente reservada para amigos cercanos o amantes.

b. **Zona personal**: es una especie de burbuja protectora, donde en la fase cercana de 45 a 75 cm se puede coger o retener a la persona (amigos íntimos, conversaciones personales). La fase lejana está aproximadamente a la distancia del brazo, de 75 a 120 cm, donde se tratan asuntos de interés y relación personal, los rasgos faciales son visibles, la voz moderada y no se percibe el olor corporal. Esta es la zona de confort utilizada generalmente con los amigos.

c. **Zona social**: en la fase cercana que se encuentra entre 120 cm y 2 m se tratan asuntos impersonales o de interacción limitada, como en el trabajo y las reuniones sociales informales. En la fase lejana de 2 a $3,5$ m la visión del individuo es completa y distante, lo que permite mantener el contacto visual.

d. **Zona pública**: en la fase cercana, que se delimita entre valores de $3,5$ a $7,5$ m, la voz es alta y hay más cuidado en las formas de expresión, las personas se observan de forma general, se contempla una visión periférica de otras personas y el contacto generalmente es superficial. Desde la fase lejana, de aproximadamente 9 m, se observa a los personajes públicos, artistas, oradores, etc.

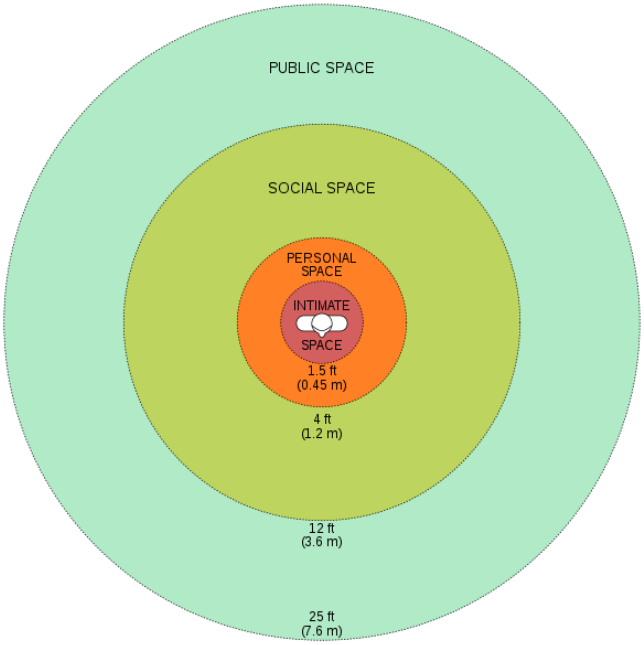


Figura 1: Representación de las cuatro zonas del espacio personal [2].

De entre las funciones psicológicas más importantes del espacio personal destacan las dos siguientes [50]:

- **La autoprotección**: funciona como amortiguador contra amenazas físicas y emocionales. Factores como las características del individuo invasor del espacio personal, cómo este se aproxime, la ansiedad o el miedo amplían la zona de espacio personal.

- **La atracción interpersonal**: el espacio personal tiene la función sociopsicológica de regular las manifestaciones de atracción interpersonal como la amistad, la atracción, la aversión o la afinidad de la personalidad.

No obstante, el espacio personal puede variar de acuerdo a la nacionalidad, la cultura, la edad, el contexto e incluso la personalidad de la persona en concreto [51,52]. Conforme el individuo crece y se desarrolla, evoluciona su sentido del espacio personal y la forma en que lo utiliza en sus relaciones sociales [53–55].

Con todo esto, la gestión humana del espacio se plantea como un sistema dinámico muy complejo que involucra factores muy concretos para cada caso estudiado: número de personas del entorno, tipo de relación con ellas, tipo de interacción que se está llevando a cabo, elementos con los que se está interactuando, etc. Con todo esto, el contexto del individuo se plantea como un pilar fundamental en las situaciones sociales de los humanos y en las tareas que los robots sociales realizan.

Cabe destacar que mientras que los humanos aprenden las convenciones sociales a lo largo de muchos años de interacción social, los robots no disponen del mismo tiempo para aprenderlas, de tal forma que un correcto diseño y configuración son básicos para lograr el futuro éxito en aplicaciones sociales.

2.3. Acompañamiento con robots sociales

Una de las aplicaciones con mayor proyección en los últimos años en el campo de los robots sociales es el acompañamiento de personas. En la sociedad existen una gran variedad de situaciones en las que un robot social puede llevar a cabo tareas de acompañamiento, ya sea acompañando a un grupo de humanos o a una sola persona. En concreto, la capacidad de realizar un acompañamiento al lado de una sola persona es una herramienta importante que todo robot urbano debería tener, respondiendo a la necesidad básica de acompañar personas de forma natural y segura. Algunas de las situaciones donde el acompañamiento lado a lado se plantea como un pilar fundamental surgen a la hora de guiar [17, 22, 56] o al acompañar personas andando [30, 57], especialmente en entornos urbanos [32] o aéreos [58].

El acompañamiento de personas es un campo que abarca una amplia variedad de disciplinas de la robótica, donde intervienen una mezcla de materias como la percepción, la navegación y la interacción humano-robot (HRI, del inglés *Human-robot interaction*). A pesar de la heterogeneidad de las materias implicadas, la tarea del acompañamiento debe ser estudiada de manera holística, lo que la convierte en una tarea especialmente complicada [32].

Concretamente, la investigación en HRI en el campo de la navegación de robots durante el acompañamiento a humanos es todavía nueva en comparación con otros campos tradicionales de la robótica, como la navegación robótica tradicional, donde los robots navegan de una manera más segura y parecida a la humana. Este campo se presenta como un reto desafiante al tener el robot que navegar de forma segura y natural mientras acompaña a una persona. Además, el robot tiene que adaptar su comportamiento para evitar colisiones con obstáculos y no molestar a otras personas del entorno. No obstante, los algoritmos de acompañamiento están evolucionando cada vez más hacia comportamientos más seguros y realistas [59].

Actualmente, existen modelos que cumplen con los objetivos propuestos anteriormente y permiten a un robot navegar con conciencia social y acompañar lado a lado a una persona en áreas urbanas concurridas [32, 59, 60], como por ejemplo se muestra en la Figura 2, incluso sin conocer el destino de la persona acompañada [57]. Aun así, no todos los modelos existentes son capaces de ajustar ciertos parámetros, como la distancia social o la velocidad de acompañamiento, en función de las características concretas del acompañamiento o del entorno particular de la persona acompañada [30].

No obstante, la impredecibilidad y complejidad de los entornos y dinámicas de los humanos hacen necesario el diseño de una gran variedad de tareas que el robot tiene que llevar a cabo durante los procesos de acompañamiento. De este modo, cuando un robot tiene como objetivo acompañar a un humano o grupo de humanos deberá ejecutar varias tareas complejas simultáneamente, como por ejemplo:

- Predecir el movimiento de personas y sus destinos finales.
- Detectar y predecir la posición de las personas acompañadas.
- Adaptar su velocidad a la velocidad de la persona o grupo (acelerando, desacelerando e incluso deteniéndose cuando es necesario).
- Adaptar su distancia respecto a la persona o grupo (acercándose o alejándose).

- Facilitar la navegación del acompañante y de otras personas del entorno.
- No invadir el espacio personal de otras personas del entorno.
- No colisionar con obstáculos estáticos y dinámicos.

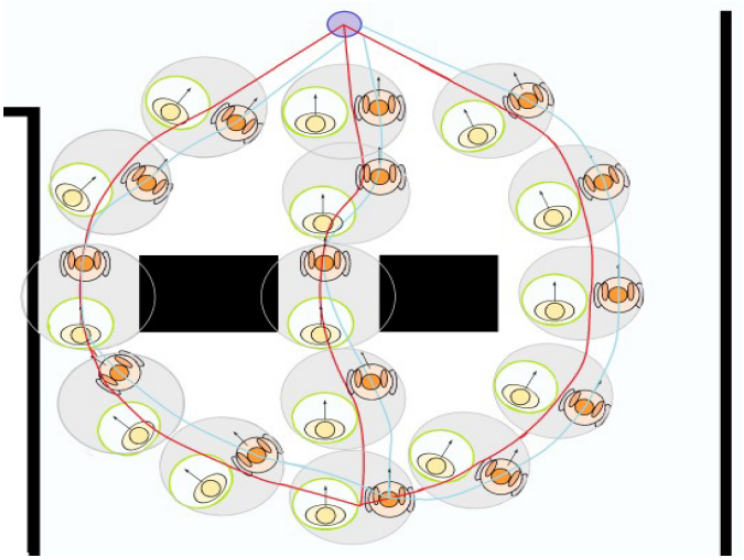


Figura 2: Representación del método para el acompañamiento de una persona por parte de un robot desarrollado en [3].

Por lo tanto, para poder navegar socialmente el robot necesita conocer las destinaciones más probables y las posiciones predichas de todas las personas de su entorno para facilitar su navegación durante el acompañamiento. Además, el robot necesita planificar todos los posibles caminos de las personas acompañadas para desarrollar un comportamiento anticipatorio y adaptativo en el acompañamiento. Después de todos los procesos necesarios, el robot debe seleccionar la mejor trayectoria que permita realizar el acompañamiento, mientras evita obstáculos estáticos y dinámicos.

De esta forma, el acompañante robot debe ser capaz de comportarse de manera similar a la que los humanos acompañan a otros humanos y, además, tiene que navegar con conciencia social y adaptándose a las personas en las diferentes formaciones posibles que adopten durante el acompañamiento.

Además, se ha demostrado que existen diversos factores que afectan a las interacciones sociales entre el robot y la persona acompañada, como, por ejemplo, si las personas acompañadas tienen experiencia en el cuidado de mascotas o en el mundo de los robots, éstas disminuyen su espacio personal con respecto al robot en comparación al espacio que las personas sin experiencia mantienen [33].

Como se ha comentado en los apartados anteriores, en la sociedad actual existen una gran variedad de situaciones en las que un robot social podría llevar a cabo tareas de acompañamiento a una persona: hacer de guía en un espacio desconocido [17, 56], ayudar a transportar objetos [23], ayudar a sanitarios a llevar medicinas en espacios delicados o vulnerables para otros humanos, etc.

2.4. Clasificación de la relación social entre personas

Un robot podría acompañar a una persona objetivo e incluso llegar a reemplazar a una persona de una pareja de humanos en tareas en las que sean necesarias dos personas, por ejemplo, para ahorrar trabajo a una de las personas o minimizar costes, riesgos o tiempo. De esta forma, se espera que el robot acompañante actúe de forma lo más cercana posible a como actuaría una persona, es decir, siguiendo las mismas convenciones sociales que seguiría la persona y adaptándose al comportamiento mostrado por la persona acompañada. Por lo tanto, el robot tiene que aprender de algún modo a comportarse de la forma más humana posible y para ello pueden ser clave los estudios realizados sobre la clasificación de la relación social entre los miembros de una pareja que se acompañan mutuamente [1].

Para facilitar la adaptación del robot al acompañamiento esperado por la persona acompañada, es importante intentar relacionar las características de los procesos de acompañamiento entre dos personas (velocidades relativas de las personas, distancia entre ellas, velocidad absoluta del grupo, etc.) con la relación social que esas dos personas tienen entre sí, ya sea más cercana o más distante, de amistad o de trabajo u otras clasificaciones posibles.

De esta forma, además, la persona acompañada podría escoger, previamente al proceso de acompañamiento, qué tipo de relación desea que el robot tenga con ella, por ejemplo, como si ambos tuvieran una relación cercana o, por el contrario, que actúe como se esperaría de un compañero de trabajo.

En general, existen una amplia variedad de estudios enfocados a la clasificación de la relación social entre personas con el objetivo de diseñar máquinas y robots capaces de interactuar socialmente con los humanos [61–63]. Acotando a entornos como las áreas urbanas, también se encuentran estudios que se centran en clasificar la relación social existente entre los miembros de grupos de peatones en espacios públicos [64] o en analizar cómo el tipo de relación entre los miembros del mismo grupo social afecta a la dinámica del grupo al moverse [65].

Por ejemplo, en [1] se presenta un estudio que se centra en analizar los grupos sociales de peatones en espacios públicos con el objetivo de identificar el tipo de relación entre los miembros del grupo. El estudio, en concreto, utiliza como grupo social de referencia a las parejas de peatones.

Aunque existen diversas clasificaciones de las relaciones sociales entre humanos [66–68], en el estudio anterior [1] se clasifica la relación de las parejas de peatones en cuatro categorías: colegas, pareja, familia o amigos [69]. Utilizando dos métodos distintos, el estudio consigue obtener relativamente buenos resultados al clasificar la relación de las parejas y distinguir entre las cuatro categorías anteriores.

A diferencia de los estudios anteriores, en este proyecto se desarrollan distintos modelos que realizan la clasificación de la relación social existente en parejas durante el acompañamiento basados en la aplicación de métodos de *clustering* y *deep learning*. Por lo tanto, los métodos implementados permiten clasificar la relación existente entre los miembros de una pareja mientras estos realizan un proceso de acompañamiento en cuatro posibles categorías, dependiendo de si su relación social es de colegas, pareja, familia o amistad. Concretamente, en los métodos basados en *deep learning* se implementan diversos modelos que utilizan redes neuronales, tanto estándares como recurrentes, para realizar la clasificación.

3. Métodos de clasificación

Para que los robots sociales encuentren su lugar en la sociedad actual en el campo del acompañamiento de personas, es de vital importancia que al realizar dichas tareas de acompañamiento sean capaces de adaptarse al comportamiento del humano. En este sentido, una parte indispensable de estas tareas consiste en observar dicho comportamiento e identificar qué tipo de relación se establece con la persona acompañada. Previamente, es necesario poder clasificar los tipos de relaciones que los humanos presentan entre parejas o grupos de dos personas. Para poder desarrollar un modelo que permita clasificar las relaciones sociales entre humanos se debe indagar en los diversos métodos que permitan cumplir con dicha función, por ejemplo: varios métodos de *clustering* y *deep learning*.

En esta sección, se describen los objetivos perseguidos al usar los métodos de clasificación de datos en la subsección 3.1, así como también se explica la base de datos utilizada, en la subsección 3.1.1. Continua con los dos grandes métodos usados para obtener la clasificación de comportamientos humanos durante el acompañamiento: *Clustering*, subsección 3.2, y *Deep Learning*, subsección 3.3. La sección dedicada al *clustering* incluye la explicación de las diferentes técnicas aplicadas en este proyecto: *K-Means* 3.2.1, *Agglomerative Hierarchical Clustering* 3.2.2, *Spectral Clustering* 3.2.3, *Mean-Shift* 3.2.4 y *Affinity Propagation* 3.2.5. La sección dedicada al *deep learning* 3.3 incluye la explicación de los diferentes métodos utilizados durante esta tesis: Redes Neuronales 3.3.1 y Redes Neuronales Recurrentes 3.3.2.

3.1. Objetivos de la clasificación

El objetivo de la tesis es desarrollar un método efectivo de clasificación de la relación de las parejas de peatones. Igual que en [1], se consideran cuatro posibles tipos de relaciones sociales que pueden darse en una pareja de peatones: colegas, pareja, familia o amistad.

Por lo tanto, la tarea principal a realizar es diseñar un programa capaz de clasificar con la mejor precisión posible los ejemplos de una base de datos de parejas llevando a cabo procesos de acompañamiento en alguna de las cuatro categorías anteriores. Tomando como referencia los resultados obtenidos en [1] al realizar la tarea descrita, se plantea utilizar métodos de clasificación diferentes a los del estudio anterior con el objetivo de mejorar los resultados del proceso de clasificación.

Si se consigue obtener un método de clasificación efectivo, este podría ser implementado en un robot social, aunque dicha tarea queda fuera del alcance de esta tesis. Si este hipotético robot social realizara un proceso de acompañamiento con una persona, este debería ser capaz de actuar como miembro activo de esa pareja, navegando a su lado y clasificando en tiempo real el comportamiento de la persona acompañada en base a las mediciones de sus sensores para poder adaptar su navegación a la relación observada que presentara la persona acompañada con él.

3.1.1. Explicación de la base de datos

Para poder desarrollar el método de clasificación deseado es necesaria una base de datos sobre la que trabajar y que poder utilizar para comprobar la eficacia de los métodos realizados. La base de datos utilizada en esta tesis fue proporcionada por Dr.Francesco Zanlungo del *Intelligent Robotics and Communication Laboratory* del ATR (*Advanced Telecommunications Research Institute*) de Kyoto, autor de varios estudios en el campo de HRI y los robots sociales [1,64,65]. Dicha base de datos contiene las lecturas de diferentes variables tomadas de parejas realizando un proceso de acompañamiento desplazándose en un entorno urbano y las etiquetas correspondientes a las relaciones sociales de las parejas. Los detalles del proceso de obtención y clasificación de los datos se pueden leer en [1] y [70].

La base de datos proporcionada presenta los ejemplos de parejas divididos en cuatro carpetas, cada una de las cuatro representando una de las categorías de la clasificación (colegas, pareja, familia o amistad). En total, la base de datos proporcionada consta de 867 ejemplos de parejas. Cada carpeta contiene los archivos de los experimentos, donde cada archivo contiene las lecturas de una única pareja. Cada archivo presenta en las primeras líneas detalles de la pareja (identificadores de cada persona del sistema de seguimiento, si son hombre o mujer, la edad y la altura).

A continuación, cada línea del archivo de datos presenta la lectura de 13 variables diferentes: tiempo, posición X del peatón 1, posición Y del peatón 1, posición Z del peatón 1, velocidad X del peatón 1, velocidad Y del peatón 1, velocidad total del peatón 1, posición X del peatón 2, posición Y del peatón 2, posición Z del peatón 2, velocidad X del peatón 2, velocidad Y del peatón 2 y velocidad total del peatón 2. Dichas variables sirven para caracterizar el tipo de acompañamiento realizado. Como los procesos de acompañamiento son procesos dinámicos dados durante la navegación, los archivos tienen diferentes mediciones al tener cada proceso una duración diferente o al salirse la pareja fuera del alcance de los sistemas de medición.

El primer paso a realizar es adecuar la base de datos a las necesidades concretas del estudio. Como el acompañamiento entre parejas es un proceso dinámico a lo largo del tiempo, la lectura de variables en un instante determinado puede no representar la realidad de la relación de manera precisa. Para esto, se realiza un programa de extracción de datos que coge las lecturas de cada experimento y realiza la media de cada parámetro, convirtiendo las lecturas de un experimento en una única lectura. Por ejemplo, si para una pareja se habían realizado lecturas cada $0,5$ segundos a lo largo de 30 segundos, se realiza la media de todas las lecturas de cada uno de los 13 parámetros, exceptuando el parámetro del tiempo que se obtiene restando los tiempos final e inicial, obteniendo como resultado un único valor para cada uno de los 13 parámetros por cada experimento y, en consecuencia, una nueva base de datos derivada de la anterior.

Además de los 13 parámetros mencionados, se calculan tres nuevas variables utilizando los parámetros conocidos debido a la importancia que demuestran tener en los procesos de acompañamiento y navegación social [1,64,65]. Las nuevas variables son las siguientes:

- Distancia entre peatones, calculada como

$$Dist = ||p_2 - p_1||, \tag{1}$$

  donde p_2 es la posición del peatón 2 y p_1 es la posición del peatón 1.

- Velocidad relativa entre peatones, calculada como

$$Vel_{Relativa} = |V_1^T - V_2^T|, \tag{2}$$

donde V_1^T es la velocidad total del peatón 1 y V_2^T es la velocidad total del peatón 2.

- Velocidad total de la pareja, calculada como

$$Vel_{Total} = \frac{V_1^T + V_2^T}{2}, \tag{3}$$

donde V_1^T es la velocidad total del peatón 1 y V_2^T es la velocidad total del peatón 2.

En resumen, la base de datos utilizada está formada por 867 ejemplos de parejas, cada uno de los cuales está definido por 16 parámetros que permitirán al método desarrollado clasificar cada ejemplo en una de las categorías definidas. Concretamente, la Tabla 1 muestra la distribución de los 867 ejemplos de parejas de la base de datos en las cuatro categorías estudiadas.

Categoría	**Nº de ejemplos**
Colegas	267
Pareja	96
Familia	218
Amistad	286
Total	867

Tabla 1: Distribución de los ejemplos de la base de datos en las cuatro categorías estudiadas.

Es importante mencionar que el programa de extracción de datos se tiene que ir adaptando para ajustar la nueva base de datos en función de las necesidades de la entrada de datos de los métodos de clasificación estudiados. En la Figura 3 se muestra el esquema general del proceso de clasificación que se llevará a cabo, donde los métodos de clasificación tienen como entrada las lecturas de los 16 parámetros de cada ejemplo de la base de datos y como salida la relación social establecida para ese ejemplo.

3.2. *Clustering*

El primer método de clasificación estudiado es el *clustering*. El *clustering* es uno de los métodos básicos más utilizados para encontrar similitudes entre los elementos que forman parte de un conjunto de datos [71], teniendo como objetivo principal separar un conjunto de datos finito y sin clasificar en diversos subconjuntos.

Por lo tanto, utilizando el método del *clustering*, un conjunto de datos se divide en varios subgrupos más o menos homogéneos utilizando como criterio para la división una medida de similitud, de modo que la similitud entre los elementos pertenecientes al mismo subgrupo es mayor que la similitud con respecto a los elementos pertenecientes a diferentes subgrupos [72].

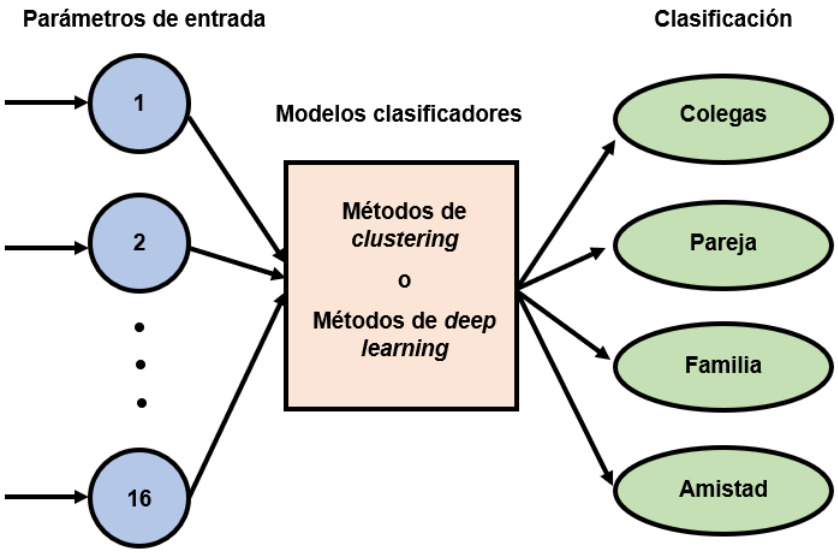


Figura 3: Esquema general del proceso de clasificación de relaciones.

De esta forma, el *clustering* puede ayudar a revelar las características particulares de cualquier estructura o patrón presente en el conjunto de datos estudiado [73].

La medida de la similitud se define en base a ciertas características (variables), que pueden ser cuantitativas (distancia entre elementos, velocidad de un objeto, duración de una determinada actividad, etc.) o cualitativas (evaluación de las relaciones entre personas, características cualitativas de productos, etc.) [74].

Los subgrupos creados en el proceso reciben el nombre de *clusters*. Es decir, los algoritmos de *clustering* asignan los elementos de un conjunto de datos (mediciones, unidades, observaciones, entidades, etc.) a un cierto número de *clusters* (grupos, subconjuntos o categorías) [75]. Por lo tanto, al finalizar el proceso de *clustering* los elementos asignados al mismo *cluster* deben ser similares entre sí, mientras que los elementos en diferentes *clusters* deben ser diferentes entre sí. En la Figura 4 se observa un ejemplo de clasificación de un conjunto de datos utilizando algún método de *clustering*, donde los datos son agrupados en tres *clusters* diferentes.

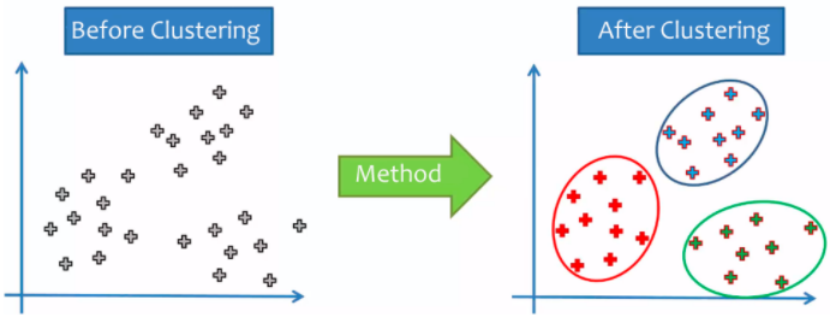


Figura 4: Ejemplo de clasificación utilizando *clustering* [4].

Debido al amplio significado que puede llegar a tomar el término *cluster*, existen varias definiciones posibles para este. Por ejemplo, en [73] se recogen tres posibles definiciones:

- *"Un cluster es un conjunto de entidades que son similares, y entidades de diferentes clusters no son similares"*.
- Un *cluster* es *"un agregado de puntos en el espacio de prueba tal que la distancia entre dos puntos cualesquiera en el cluster es menor que la distancia entre cualquier punto en el cluster y cualquier punto que no esté en él"*.
- *"Los clusters pueden describirse como regiones continuas de este espacio (espacio de características d-dimensional) que contiene una densidad relativamente alta de puntos, separados de otras regiones similares por regiones que contienen una densidad relativamente baja de puntos"*.

Como el concepto de *cluster* no presenta una definición precisa y universal, a lo largo de los años se han desarrollado una gran variedad de métodos de *clustering* [76]. No obstante, tanto la utilización de diferentes métodos de *clustering* como la utilización del mismo método, pero variando alguno de sus parámetros, puede ocasionar resultados completamente diferentes. Es decir, diferentes criterios de *clustering* pueden asignar a un individuo especifico en grupos muy diferentes y producir por lo tanto diferentes clasificaciones. Aun así, no existe ninguna manera de determinar a priori qué criterio es el mejor en general [75].

Por lo tanto, es necesaria la utilización de diferentes métodos de *clustering* para poder encontrar el algoritmo que realice una mejor clasificación de la base de datos estudiada. Dichos métodos se explican en las siguientes secciones.

3.2.1. *K-Means*

El primer método de *clustering* implementado es el conocido como *K-Means*. Este método es el más simple y común de los algoritmos usados en el *clustering*. Además, es uno de los métodos utilizados con más éxito en conjuntos de datos de gran tamaño, debido a su simplicidad y a su atractivo poco coste computacional [77].

El objetivo del método es asignar los elementos del conjunto de datos estudiado en k clusters, representados por sus centros o medias (centroides), optimizando alguna función objetivo tomada como criterio de referencia. La función a optimizar más utilizada es la distancia euclidiana entre el punto objetivo estudiado y cada uno de los centros de los *clusters*. De esta forma, cada elemento del conjunto de datos se asigna al *cluster* que presente la distancia euclidiana mínima respecto a su centroide.

En este algoritmo, el conjunto de datos estudiado se clasifica en un número de *clusters* k definido por el usuario. La idea principal es definir k centroides, uno para cada *cluster*. De esta forma, la función objetivo J a optimizar es:

$$Minimizar J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^j - c_j \right\|^2, \tag{4}$$

donde $\left\| x_i^j - c_j \right\|^2$ es la distancia euclidiana entre un punto del conjunto de datos estudiado x_i^j y el centro del *cluster* c_j [78]. En la ecuación se recorren todos los puntos del conjunto de datos

estudiado, n, y se asignan a los k *clusters* deseados.

Inicialmente, el algoritmo empieza con un conjunto inicial de centros de *clusters*, escogidos al azar o de acuerdo con algún procedimiento heurístico. En cada iteración, cada elemento es asignado al *cluster* cuyo centro se sitúe a la distancia mínima según la ecuación 4 y los centros de los *clusters* son recalculados. En este sentido, el centro de cada *cluster* se calcula como la media de todos los elementos pertenecientes a ese *cluster*:

$$\mu_k = \frac{1}{N_k} \sum_{q=1}^{N_k} x_q, \tag{5}$$

donde N_k es el número de muestras pertenecientes al *cluster* k, los puntos del conjunto de datos estudiado asignados al *cluster* son x_q, y μ_k es la media obtenida para el *cluster* k [76].

3.2.2. *Agglomerative hierarchical clustering*

El método del *agglomerative hierarchical clustering* construye los *clusters* fusionando recursivamente los elementos del conjunto de datos estudiado y los posteriores subconjuntos creados. Inicialmente, cada elemento del conjunto de datos representa un *cluster* por sí mismo. A continuación, los *clusters* son fusionados sucesivamente en *clusters* más grandes hasta que se cumplen ciertas condiciones de finalización y se obtiene la estructura de *clusters* deseada. El proceso de fusionar *clusters* se realiza de acuerdo con alguna medida de similitud escogida con el objetivo de optimizar algún criterio.

El *hierarchical clustering* es un método opuesto al método *K-Means*, donde los datos se asignan en k *clusters* sin ninguna estructura jerárquica con el objetivo de optimizar una función utilizada como referencia [78].

3.2.3. *Spectral clustering*

El método del *spectral clustering* consiste en algoritmos que clasifican los elementos de un conjunto de datos utilizando vectores propios de matrices derivadas a partir de los datos. De esta forma, estos algoritmos se basan en la estructura propia de una matriz de similitud para separar los elementos del conjunto de datos estudiado en diferentes *clusters*, de forma que elementos del mismo *cluster* son más similares entre sí que elementos de diferentes *clusters*.

Una de las características que justifica el amplio uso en la actualidad del *spectral clustering* es el hecho de que no realiza suposiciones sólidas en la posible forma de los *clusters*. De esta forma, el *spectral clustering* es capaz de resolver problemas muy generales, a diferencia, por ejemplo, del método *K-Means*, donde los *clusters* resultantes forman conjuntos convexos. Además, el método puede ser implementado de manera eficiente incluso para grandes conjuntos de datos [78].

3.2.4. *Mean-Shift*

La idea básica del método de *clustering Mean-Shift* es encontrar regiones del espacio de características del conjunto de datos estudiado que contengan una gran densidad de puntos. Para realizarlo, en cada punto del conjunto de datos se ejecuta una iteración donde se calcula la media de los puntos a su alrededor con el objetivo de localizar los máximos locales de densidad para que estos definan los diferentes *clusters*. De esta forma, todos los puntos que convergen en el mismo máximo local pertenecen al mismo *cluster* [79,80].

En contraste con otros método de *clustering*, como por ejemplo el *K-Means*, en el método de *Mean-Shift* no se realizan suposiciones sobre la forma de la distribución de los datos ni del número de *clusters*. De esta forma, este método permite que no sea necesario especificar el número de *clusters* buscado previamente a ejecutarse.

A pesar de que los algoritmos *Mean-Shift* son computacionalmente lentos, se pueden acelerar y escalar a grandes conjuntos de datos. Además, son especialmente utilizados con los conjuntos de datos de baja dimensión que tienen *clusters* no convexos, como por ejemplo en tareas de segmentación de imágenes [79].

3.2.5. *Affinity Propagation*

El método de la *Affinity Propagation* es un algoritmo de *clustering* relativamente nuevo introducido por [81]. Este método determina un llamado ejemplar para cada *cluster*, definido como la muestra que es más representativa para ese *cluster*. Dichos ejemplares se eligen entre las muestras de datos observados, y no son calculados como promedios hipotéticos de muestras de *clusters*, como en otros métodos de *clustering* [82]. Además, este método también puede ejecutarse sin definir el número de *clusters* buscado.

El método de la *Affinity Propagation* funciona tomando como entrada medidas de similitud entre pares de puntos de datos y considerando simultáneamente a todos los puntos de datos como potenciales centros de *clusters*, los conocidos como ejemplares. Para encontrar ejemplares apropiados, el método acumula evidencias de "responsabilidad" $R(i,k)$ del punto de datos i sobre qué tan adecuado es el punto k para servir como ejemplar para el punto i, y acumula evidencias de "disponibilidad" $A(i,k)$ del punto a candidato ejemplar k para saber qué tan apropiado sería para el punto i que se eligiera el punto k como su ejemplar. En consecuencia, cuanto mayor sea $R(:,k) + A(:,K)$, mayor es la probabilidad de que el punto k sea el centro del *cluster* final. Por lo tanto, este método busca *clusters* a través de un proceso iterativo hasta que surge un conjunto de ejemplares de alta calidad y los *clusters* correspondientes emergen. En el proceso iterativo, los ejemplares identificados comienzan desde su número máximo hasta que se obtienen m ejemplares y no se modifican más, o hasta que el algoritmo de *Affinity Propagation* converge. Los m *clusters* encontrados basados en m ejemplares son la solución del método de *clustering* de la *Affinity Propagation* [81,83].

Para ilustrar la diferencia entre los diferentes métodos de *clustering*, en la Figura 5 se puede ver el resultado de aplicar los cinco métodos explicados anteriormente sobre dos conjuntos de datos 2D. Como se observa, los resultados de la clasificación varían en función del método de *clustering* utilizado.

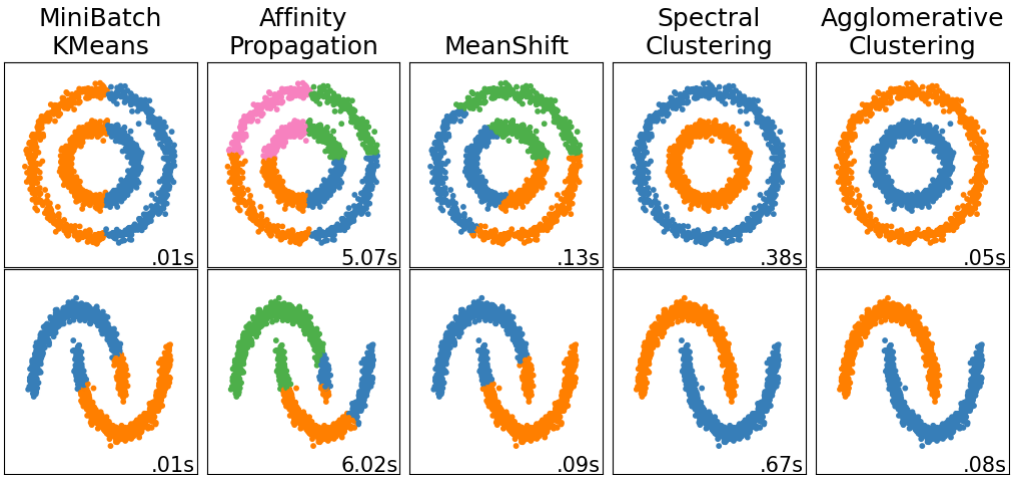


Figura 5: Resultados de la clasificación realizada por los métodos de *clustering* explicados sobre dos conjuntos de datos 2D [5].

3.3. Deep Learning

Una vez estudiados los diferentes métodos de *clustering* se decide investigar un moderno campo de la ciencia conocido como *deep learning* que presenta novedosos y potentes modelos de clasificación.

A lo largo de la historia, científicos, inventores e ingenieros han soñado con la idea de desarrollar máquinas capaces de pensar, aprender y decidir por si mismas. Hoy en día, la inteligencia artificial (IA) se postula como uno de los campos científicos con mayor proyección y prosperidad, tanto en aplicaciones mundanas como en la investigación y tecnologías más avanzadas, como por ejemplo reconocer y comprender el habla [84], la escritura [85] o las imágenes [86], automatizar tareas rutinarias [87], diseñar videojuegos más sofisticados y complejos [88] o ayudar a diagnosticar en medicina [89].

En los albores de la IA, se abordaron básicamente problemas que pueden describirse utilizando un conjunto de normas o reglas matemáticas, considerados intelectualmente difíciles para los seres humanos, pero relativamente sencillos para los ordenadores. No obstante, posteriormente, la comunidad científica comprobaría que el auténtico reto para la IA sería resolver tareas que son, en general, relativamente sencillas de realizar para los seres humanos pero complicadas de describir formalmente, es decir, tareas que las personas resuelven intuitivamente en base a su experiencia o conocimiento vital y que, en parte, realizan de manera casi automática, como reconocer palabras en el habla o caras en imágenes o clasificar relaciones entre personas. Por lo tanto, una condición del éxito de la IA y sus aplicaciones radica en cómo trasladar ese conocimiento, en gran medida subjetivo e intuitivo y, en consecuencia, difícil de describir de manera formal, a los ordenadores y sistemas informáticos [90].

Esta dificultad en el traspaso de conocimiento humano hace patente la necesidad de los sistemas de IA de tener las habilidades para adquirir dicho conocimiento sin ser expresamente programados y a partir de los datos sin procesar obtenidos del mundo real. El conjunto de algoritmos que otorga dichas habilidades se conoce como aprendizaje automático o *machine learning*. De esta forma, la introducción del *machine learning* permite a los sistemas de IA realizar tareas y

resolver problemas para los cuales es necesario el conocimiento del mundo real.

No obstante, el rendimiento de los algoritmos de *machine learning* depende en gran medida de la representación de los datos del mundo real que se les proporciona. Cada elemento incluido en la representación de la realidad que proporciona información concreta de un aspecto del conjunto de datos analizado se conoce como característica. En este sentido, considerables tareas de IA pueden ser resueltas diseñando un extractor del conjunto correcto de características para esas tareas y, posteriormente, proporcionando esas características a un algoritmo de *machine learning*.

Sin embargo, para muchas tareas y problemas resulta complicado conocer exactamente qué características deben ser extraídas solamente a partir de un conjunto de datos sin procesar. Una solución a este problema es utilizar *machine learning* para descubrir no solo el valor deseado de la representación de la realidad a la salida, sino también la representación en sí misma. Este enfoque se conoce como aprendizaje de representación o *representation learning*. De esta forma, el *representation learning* es un conjunto de métodos que permite a un sistema de IA alimentarse con un conjunto de datos brutos sin procesar y descubrir y aprender automáticamente las representaciones necesarias que permitan la futura detección o clasificación de datos [91]. Las representaciones aprendidas presentan a menudo un mayor rendimiento que el obtenido con representaciones diseñadas a mano y permiten que los sistemas de IA se adapten rápidamente a nuevas tareas con una mínima intervención humana [90].

Actualmente, la tecnología de *machine learning* se encuentra presente en multitud de aplicaciones, tales como: filtrar contenido en redes sociales, realizar recomendaciones en sitios web de comercio electrónico, realizar búsquedas por internet, identificar objetos en imágenes, transcribir voz en texto, relacionar noticias, publicaciones o productos con los intereses de los usuarios y se encuentra cada vez más presente en productos de consumo como cámaras y teléfonos inteligentes [91].

En la misma línea, otra solución a los problemas planteados es diseñar los sistemas para que aprendan de la experiencia y comprendan el mundo en términos de una jerarquía de conceptos, con cada concepto definido en términos de su relación a conceptos más simples. Al recopilar conocimientos de la experiencia, se evita la necesidad de que las personas encargadas de diseñar los sistemas tengan que especificar explícitamente toda la información que el sistema requiere, permitiéndole aprender conceptos complicados construyéndolos a partir de conceptos más simples. Este enfoque de la IA se denomina aprendizaje profundo o *deep learning* [90]. De esta forma, el desarrollo en el campo de la IA se ha expandido con el tiempo como se muestra en la Figura 6 [92].

El *deep learning* forma parte de un conjunto más amplio de métodos de *machine learning* basados en descubrir y asimilar representaciones de conjuntos de datos. De esta forma, una observación (por ejemplo, una imagen) puede ser representada de varias formas (por ejemplo, un vector de píxeles), pero algunas representaciones facilitan el aprendizaje de ciertas tareas y la resolución de ciertos problemas (por ejemplo, "¿es esta imagen una cara humana?") sobre el conjunto de datos estudiado, teniendo la investigación en este campo como objetivos definir qué representaciones son mejores y cómo crear modelos para reconocer estas representaciones [93].

Los métodos de *deep learning* son métodos de *representation learning* con múltiples niveles de representación, obtenidos componiendo módulos simples pero no lineales, cada uno de los cua-

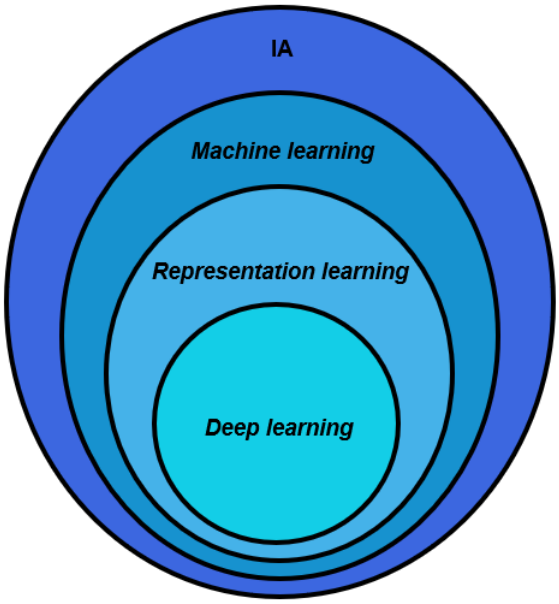


Figura 6: Diagrama de Venn de los diferentes campos que forman parte del mundo de la IA.

les transforma la representación en un nivel (comenzando con la entrada sin procesar) en una representación a un nivel superior, ligeramente más abstracto. Mediante la composición de suficientes transformaciones de este tipo, se pueden aprender funciones muy complejas. Para tareas de clasificación, las capas superiores de representación amplían aspectos o características de la entrada que son importantes para la discriminación y suprimen variaciones irrelevantes [91].

Por lo tanto, se puede definir el *deep learning* como un conjunto de algoritmos de *machine learning* que intenta modelar representaciones complejas en conjuntos de datos utilizando arquitecturas computacionales conocidas como redes neuronales que admiten transformaciones no lineales múltiples e iterativas de datos [94]. Además, resuelve problemas debidos a la complejidad de los datos al utilizar representaciones que se expresan en términos de otras representaciones más simples, permitiendo a los sistemas y ordenadores construir y definir estructuras complejas a partir de estructuras más simples.

Los enfoques de *deep learning* funcionan aprendiendo no solo a predecir, sino también a representar correctamente los datos, para que la representación obtenida sea adecuada para realizar predicciones. En este sentido, dado un conjunto de datos de entrada a estudiar y un conjunto de datos que indique la salida deseada, los enfoques de *deep learning* funcionan alimentando con los datos de entrada una red neuronal que produce transformaciones sucesivas de los datos de entrada hasta que una transformación final predice la salida deseada. Las transformaciones producidas por la red neuronal se realizan en base a los conjuntos de datos de entrada y salida dados, de modo que cada transformación facilita la relación de los datos de entrada con la clasificación de salida deseada [6].

La tecnología basada en *deep learning* ha logrado importantes avances en la resolución de problemas que se habían resistido en el campo de la IA durante años. Además, el *deep learning* ha resultado ser muy eficaz a la hora de descubrir estructuras intrincadas en datos de altas dimensiones, lo que permite su aplicación en muchos dominios, como la ciencia [95, 96], los

negocios [97] e incluso en aplicaciones gubernamentales [91] o en campos como la visión por computador, el reconocimiento de patrones o el reconocimiento del habla [98]. Además, con el paso del tiempo se ha vuelto más útil a medida que la cantidad de datos disponible ha ido aumentando gracias, por ejemplo, a las mejoras tecnológicas y a una mayor capacidad de recolección de datos. Esto ha llevado a los modelos de *deep learning* a crecer en tamaño a medida que la infraestructura informática, tanto de hardware como de software, ha mejorado, resultando en aplicaciones que permiten la resolución de tareas cada vez más complicadas con un aumento de la precisión [90].

3.3.1. Red Neuronal

Como se ha mencionado anteriormente, el *deep learning* intenta modelar representaciones complejas en conjuntos de datos utilizando arquitecturas computacionales conocidas como redes neuronales [94]. Las redes neuronales son sistemas de IA inspirados en las estructuras del cerebro humano, basándose en modelos de cómo el aprendizaje ocurre o podría ocurrir en el cerebro [90], donde miles de millones de células nerviosas llamadas neuronas se conectan y comunican entre sí y crean una red neuronal biológica que dota a los humanos de las capacidades necesarias para, por ejemplo, poder hablar, leer, moverse, respirar, comprender, resolver problemas o almacenar datos [99, 100].

Las redes neuronales destacan especialmente por su capacidad para procesar información gracias a la estructura que presentan. Estas se construyen mediante unidades de computación complejas, conocidas como neuronas, que se distribuyen en capas y se encuentran interconectadas formando una red de elementos que trabajan juntos para resolver problemas específicos [92]. Además, las redes neuronales se diseñan utilizando tres tipos de capas diferentes, cada una de las cuales puede estar diseñada con un número diferente de unidades o neuronas. Los tres posibles tipos de capas que se pueden encontrar en una red neuronal son:

- Capa de entrada: tiene como objetivo recibir los datos sin procesar que se han enviado a la red. Actúa como entrada del sistema.

- Capas ocultas: no reciben ni envían información al entorno, sino que realizan los procesamientos internos de la red. La función de estas capas está determinada por la capa de entrada, los parámetros del sistema (matrices de pesos) y la relación entre ellas y capas posteriores. Los pesos entre las unidades de la capa de entrada y las unidades de las capas ocultas (o entre capas ocultas) determinan cuándo es necesario activar una unidad de la capa oculta. Habitualmente, las redes neuronales con dos o más capas ocultas se denominan redes neuronales profundas.

- Capa de salida: proporcionan la respuesta de la red neuronal a los datos de la entrada. La función y respuesta de la capa de salida varía según el modelo estudiado, los parámetros del sistema y la conexión entre las unidades ocultas y las unidades de salida.

En la Figura 7 se puede observar un ejemplo de red neuronal, donde cada círculo representa una neurona, las flechas entrantes son las entradas de la neurona y las flechas salientes son sus las salidas. La capa situada abajo no tiene flechas entrantes porque es la capa de entrada de la red, mientras que la capa de arriba no tiene flechas de salida debido a que es la capa de salida. Las capas restantes son las capas ocultas de la red y se encuentran conectadas con la capa de

entrada y la capa de salida. Se puede observar que esta red neuronal (ver Figura 7) tiene cuatro unidades en la capa de entrada, seis unidades en la primera capa oculta, cinco unidades en la segunda capa oculta y tres unidades en la capa de salida.

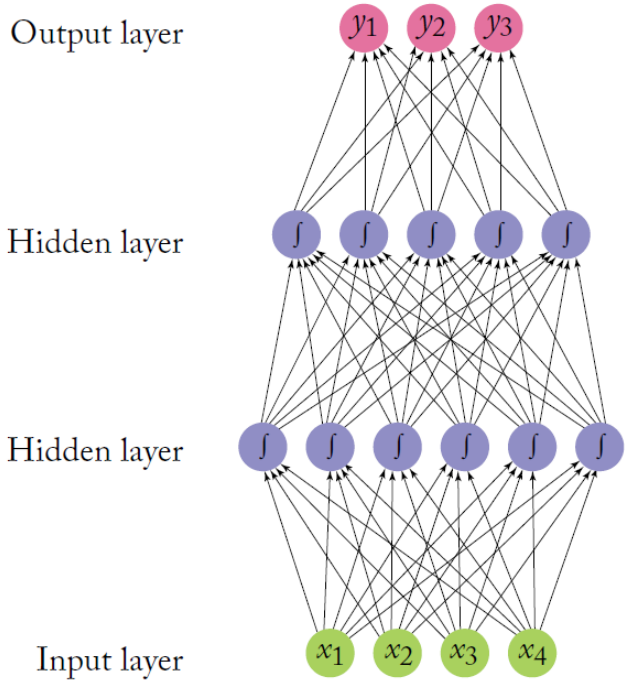


Figura 7: Representación de un ejemplo de red neuronal [6].

En las redes neuronales, cada unidad de una capa se relaciona con las unidades de la capa siguiente. El procesamiento de la información en la red implica la entrada de datos desde las unidades de la capa de entrada y pasar a través de la red fluyendo de una capa a otra hasta llegar a las unidades de la capa de salida. En el caso de las redes neuronales más sencillas, la información es transmitida sólo en una dirección, es decir, de las unidades de entrada, a las unidades ocultas, si hay, y luego a las unidades de salida [92].

Las redes neuronales tienen la capacidad de descubrir estructuras intrincadas en grandes conjuntos de datos al dividir la representación deseada en una serie de representaciones simples anidadas, cada una descrita por una capa diferente de la red neuronal. Los datos a estudiar se presentan en la capa de entrada, que contiene las variables que podemos observar. Luego, una serie de capas ocultas extraen características cada vez más abstractas de la representación. Aunque los valores de las capas ocultas no se dan en los datos, el modelo de la representación debe determinar qué características son útiles para explicar las relaciones en el conjunto de datos estudiado [90]. Posteriormente, se indica al sistema cómo debe cambiar sus parámetros internos, que se utilizan para calcular la representación en cada capa a partir de la representación en la capa anterior [91].

El potencial de las redes neurales radica en su capacidad de aprender a través del entrenamiento. Durante el proceso de entrenamiento, la red neuronal es alimentada continuamente con ejemplos del conjunto de datos estudiado para los que se conoce el resultado, por ejemplo, de su clasificación o predicción deseada. A continuación, las respuestas que proporciona la red a los ejemplos de entrada se comparan con los resultados conocidos. Entonces, la información pro-

cedente de esta comparación se envía a través de la red y cambia los parámetros del sistema gradualmente. A medida que el entrenamiento avanza y la red es entrenada con más datos, la red neuronal se va haciendo cada vez más precisa en la replicación de resultados conocidos. Una vez finalizado el proceso de entrenamiento, la red entrenada se puede aplicar a casos futuros con nuevos ejemplos y datos en los que se desconoce el resultado con el objetivo de clasificar o predecir dichos casos.

En consecuencia, para entrenar una red neuronal es necesario dividir el conjunto de datos de entrada en dos grupos: *training set* y *test set*. De un lado, los datos del *training set* sirven para entrenar la red neuronal con el objetivo de ajustar los parámetros del sistema y obtener la mejor representación o modelo posible. De otro lado, los datos del *test set* sirven para comprobar si la representación o modelo obtenido a partir del *training set* refleja la realidad y complejidad de las relaciones y características de la totalidad del conjunto de datos de entrada.

A continuación, se presenta más en detalle el proceso que lleva a cabo la red neuronal y la notación matemática utilizada. Para empezar, se presenta la red neuronal más simple, conocida como perceptrón, que se puede describir mediante el siguiente modelo lineal:

$$RN(x) = xW + b \tag{6}$$

donde W es la matriz de pesos y b el término de sesgo. No obstante, para ir más allá de funciones lineares y poder describir modelos más complejos, se introducen capas ocultas no lineales. De esta forma, una red neuronal con una sola capa oculta se puede describir utilizando la siguiente ecuación:

$$RN(x) = g(xW^1 + b^1)W^2 + b^2 \tag{7}$$

donde W^1 y b^1 son la matriz de pesos y el término de sesgo de la primera transformación lineal de la entrada, g es una función no lineal llamada función de activación y W^2 y b^2 son la matriz de pesos y el término de sesgo de la segunda transformación lineal.

Las matrices de pesos y los términos de sesgo que definen las transformaciones lineales son los parámetros de la red neuronal. Conjuntamente con la entrada, los parámetros determinan la salida de la red. El algoritmo de entrenamiento es el responsable de configurar los valores de los parámetros para conseguir que las predicciones de la red sean correctas. Además, la no linealidad de la función de activación g tiene un papel fundamental en la capacidad de la red neuronal de representar funciones complejas [6].

Existen varias alternativas a la hora de seleccionar la función de activación a utilizar en las neuronas de las diferentes capas de la red. De entre las diversas opciones, se detallan dos funciones de activación:

- Función de activación *ReLU*: una de las funciones más simples, pero que mejores resultados presenta. Se define según la ecuación

$$R(z) = max\{0, z\} \tag{8}$$

donde z es la entrada de la neurona. En la Figura 8 se puede ver la representación gráfica de esta función, donde se puede observar como a las entradas con valores inferiores a 0 ($z < 0$) se les asigna un valor de 0.

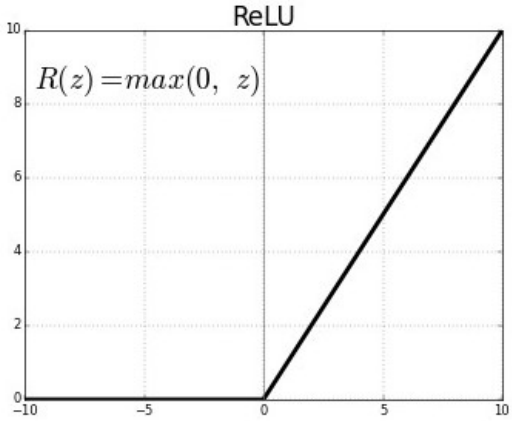


Figura 8: Representación gráfica de la función de activación ReLU [7].

- Función de activación *softmax*: esta función se basa en calcular las "evidencias" de que un determinado ejemplo del conjunto pertenezca a una de las categorías estudiadas y, posteriormente, convierte esas evidencias en probabilidades de que el ejemplo procesado pertenezca a cada una de las posibles categorías. Es decir, devuelve la distribución de probabilidad sobre categorías de salida mutuamente excluyentes. La salida de esta función es un vector de tantos componentes como posibles categorías, donde cada componente es un dígito entre 0 y 1 y la suma de todos ellos es igual a 1.

  En la Figura 9 se puede observar un ejemplo de la salida de la función *softmax*, donde se muestra que hay un $46\,\%$ de probabilidades de que el ejemplo procesado pertenezca a la categoría 1, un $34\,\%$ de que sea de la categoría 2 y un $20\,\%$ de pertenecer a la categoría 3. Por lo tanto, una buena clasificación presentará un solo componente de dicho vector con un valor cercano a 1, mientras que el resto de componentes tendrán un valor cercano a 0 [101].

Figura 9: Ejemplo de salida de la función de activación Softmax [8].

Además, cuando se entrena una red neuronal se define una función de pérdida objetivo $L(\hat{y}, y)$, que calcula la pérdida de predecir $\hat{y}$ cuando la salida verdadera es y. La pérdida $L(\hat{y}, y)$ asigna un valor numérico (un escalar) a la salida de la red $\hat{y}$ dada la salida real esperada y. Una de las funciones de pérdida más usadas es la función de entropía cruzada o *croos-entropy function*, definida en la Ecuación 9.

$$L(\hat{y}, y) = -((y \log \hat{y}) + (1 - y) \log (1 - \hat{y})) \tag{9}$$

Una vez expuestos los elementos implicados en el entrenamiento de una red neuronal, se definen las diversas etapas del proceso. En la primera etapa, conocida como *forwardpropagation*, la red recibe los datos del *training set*. Estos datos son procesados por las neuronas de las diferentes capas, donde se les aplican las transformaciones correspondientes en función de los parámetros de la red, hasta la capa de salida, donde se realiza la clasificación de los datos.

En una segunda etapa, se utiliza la función de pérdida definida para estimar la pérdida o error entre la clasificación correcta y la realizada por el modelo. El objetivo del entrenamiento es minimizar la pérdida a través de los diferentes ejemplos del conjunto de datos. Esto se consigue ajustando los parámetros de la red durante el entrenamiento para lograr un método que clasifique correctamente los ejemplos de la base de datos.

Con la pérdida calculada se puede pasar a la tercera etapa, llamada *backpropagation*, en la que la información de la pérdida se propaga desde la capa de salida de la red hasta la capa de entrada, pasando por las neuronas de las capas ocultas.

Después de que todas las neuronas hayan recibido la información de la pérdida de la fase de *backpropagation*, se ajustan los parámetros de la red neuronal con el objetivo de que la pérdida presente un valor lo más próximo posible a cero cuando se vuelva a procesar otro ejemplo para su clasificación. Para realizar esta tarea se utiliza un algoritmo conocido como descenso de gradiente o *gradient descent*. El algoritmo se encarga de ajustar los parámetros de la red en pequeños incrementos. Para esto, el algoritmo calcula la derivada de la función de pérdida objetivo para los parámetros de la red W y b para conocer cómo estos parámetros afectan al resultado de la función de pérdida y con el objetivo de acercarse al mínimo global de la función, donde se obtiene un error bajo en la salida de la red. Por lo tanto, el objetivo de este algoritmo es ajustar los parámetros del sistema para minimizar la función de pérdida para el *training set*. En la Figura 10 se puede observar una representación gráfica del algoritmo de *gradient descent*.

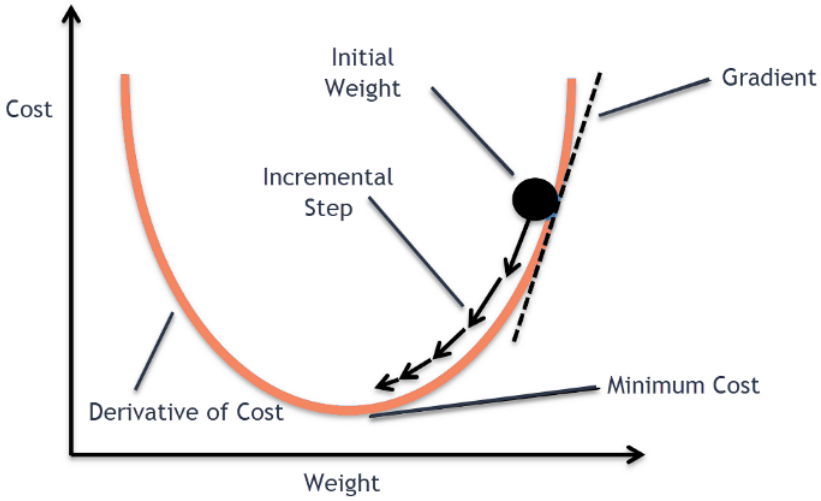


Figura 10: Representación gráfica del algoritmo de *gradient descent* [9].

En resumen, el procedimiento de entrenamiento consiste en obtener el vector de entrada a partir de los ejemplos del *training set*, procesar y clasificar el ejemplo para obtener la salida, calcular la pérdida o error, calcular el gradiente y ajustar los parámetros de la red en consecuencia. El proceso se repite para todo el conjunto de ejemplos del *training set* hasta que finaliza el proceso

de entrenamiento y, idealmente, el promedio de la función objetivo deja de disminuir. Después del entrenamiento, se calcula el rendimiento del sistema en el conjunto de ejemplos del *test set*. De esta forma, se comprueba la capacidad de generalización de la red neuronal, es decir, la capacidad que tiene para producir respuestas o predicciones correctas sobre nuevas entradas que nunca ha visto durante el proceso de entrenamiento [91].

Existen diversas variantes del algoritmo de *gradient descent* que pueden implementarse en una red neuronal. Por ejemplo, en el *Stochastic Gradient Descent* los parámetros de la red se actualizan después de calcular la pérdida para cada ejemplo del *training set*, mientras que en el *Batch Gradient Descent* los parámetros se actualizan después de procesar todos los ejemplos del *training set* utilizando la media de los gradientes. Un término medio de los dos algoritmos anteriores es el conocido como *Mini-Batch Gradient Descent*. Este algoritmo realiza la actualización de los parámetros después de procesar entero un conjunto de ejemplos de tamaño fijo, conocido como *mini-batch*, utilizando la media de los gradientes del conjunto.

Otro factor a tener en cuenta en el proceso de entrenamiento es el número de veces que el modelo procesa todo el conjunto de datos del *training set* para entrenar. Este factor se conoce como número de épocas o *epochs*. En cada ciclo (*epoch*) del entrenamiento todos los datos del *training set* son procesados por la red neuronal para que esta aprenda sobre ellos. Si también se especifica el tamaño de los *mini-batch*, cada *epoch* constará de más iteraciones internas cuanto más pequeño sea el tamaño de *mini-batch*. Por ejemplo, si se tiene un *training set* de 800 ejemplos y se configura el modelo con un tamaño de *mini-batch* de 100 y un número de *epochs* de 10, en el proceso de entrenamiento se realizarán 8 iteraciones para completar una *epoch*. Además, en cada iteración también se llevarán a cabo las fases de *forwardpropagation* y *backpropagation* y, por lo tanto, la red neuronal actualizará más veces los parámetros de la red, permitiendo entrenar al modelo para minimizar el error o pérdida.

Durante el proceso de entrenamiento existe un elemento, llamado *learning rate*, el valor del cual se plantea como un factor importante en el proceso de actualización de los parámetros de la red. De esta forma, la actualización de los parámetros W y b se realiza según las siguientes ecuaciones:

$$W := W - \alpha \frac{\partial J(W,b)}{\partial W} \tag{10}$$

$$b := b - \alpha \frac{\partial J(W,b)}{\partial b} \tag{11}$$

donde α es el *learning rate* y $\frac{\partial J(W,b)}{\partial W}$ y $\frac{\partial J(W,b)}{\partial b}$ son las derivadas parciales de la función de pérdida respecto a los parámetros W y b, respectivamente.

De esta forma, el *learning rate* tiene un efecto directo en la actualización de los parámetros de la red. Un *learning rate* con un valor demasiado grande puede evitar que el proceso converja en el mínimo global por muy rápido que se actualicen los parámetros al trabajar con grandes intervalos. Por el contrario, un valor demasiado pequeño puede conllevar que la actualización se quede atrapada en un mínimo local y que los parámetros W y b no se actualicen correctamente o que el tiempo necesario para optimizar los parámetros sea demasiado grande [6]. En la Figura 11 se pueden ver las posibles evoluciones de la pérdida o error del modelo durante el proceso de

entrenamiento a lo largo de las épocas o *epochs* en función del valor de *learning rate* seleccionado.

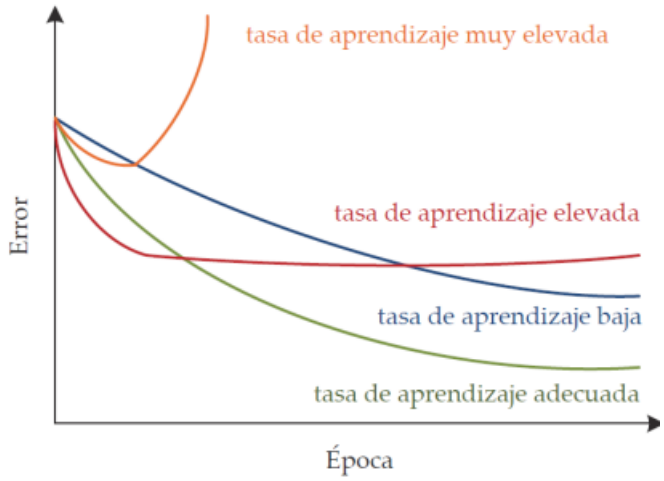


Figura 11: Representación gráfica del error del modelo durante el entrenamiento en función del *learning rate* utilizado [10].

Técnicamente, muchos de los elementos del diseño de redes neuronales explicados hasta ahora reciben el nombre de hiperparámetros, reservando el término "parámetros" únicamente para los parámetros W y b de la red. Estos hiperparámetros son aquellas variables del modelo que se deben configurar y ajustar durante el diseño y que permiten controlar el proceso de entrenamiento de la red neuronal. Algunos de los hiperparámetros utilizados en el diseño y optimización de redes neuronales son:

- Número de capas ocultas
- Número de neuronas de las capas ocultas
- Tamaño de *mini-batch*
- Número de *epochs*
- Elección de la función de activación
- Elección del *learning rate*
- Implementación de regularización

Por lo tanto, el proceso de aprendizaje trata de encontrar los parámetros de la red que hagan que esta se comporte de la manera deseada. Si los parámetros se establecen correctamente, una red neuronal con suficientes neuronas y una función de activación no lineal puede aproximar a una gama muy amplia de funciones matemáticas. Cabe destacar que proporcionar a la red un conjunto adecuado de ejemplos sobre los que estudiar y codificar los datos de entrada de una manera adecuada facilita parte del proceso de aprendizaje [6].

Con todo esto, se puede afirmar que el objetivo principal de las redes neuronales es conseguir un modelo o representación que presente la mejor precisión posible tanto en el *training set* como en el *test set*. Si el sistema presenta una buena precisión sobre el *training set* significa que el

sistema realiza una correcta representación del conjunto de datos sobre el que ha entrenado. Si el sistema también presenta una buena precisión sobre el *test set* significa que el modelo es capaz de representar de manera precisa el conjunto de datos estudiado y las relaciones entre los datos.

Algunas de las ventajas que presentan las redes neuronales sobre otros sistemas estadísticos o de IA son, por ejemplo, que los modelos de redes neuronales no realizan suposiciones sobre las propiedades o la distribución de los datos. Además, sus características en el procesamiento de información, como por ejemplo su capacidad de aprendizaje, alto paralelismo, adaptabilidad, escalabilidad, tolerancia a fallos, no linealidad, tolerancia al ruido o capacidad de generalización [92], les hacen enfrentarse a problemas complejos del mundo real. Como se ha comentado, una de las características distintivas de las redes neuronales es su capacidad para aprender de la experiencia y de los ejemplos y, además, adaptarse a situaciones cambiantes [102].

Las redes neuronales han demostrado ser un avance en el mundo de la IA, especialmente desde que recursos como las herramientas de procesamiento de información, *big data* o *Internet of Things* adquirieron una posición significativa en el mundo de la ciencia y la tecnología [99], y han logrado posicionarse como una herramienta efectiva a la hora de identificar tendencias en datos y patrones y realizar previsiones y predicciones [92].

En los últimos años, las redes neuronales se han utilizado en multitud de disciplinas para una gran variedad de tareas. Campos como por ejemplo ciberseguridad [96], biología [103], política [104], agricultura [105, 106], militar [107] o la economía [97, 108–110] han visto como las aplicaciones utilizando redes neuronales son cada vez más abundantes, destacando especialmente su uso en el mundo de la robótica [86] y la medicina [89, 99, 111–113].

Al finalizar el proceso de entrenamiento, se puede obtener una red neural que presenta un modelo que captura demasiadas particularidades de los datos del *training set* y no se ajusta a los nuevos datos del *test set*. Es decir, el modelo obtenido clasifica correctamente los datos de entrenamiento, pero tiene dificultades para clasificar los datos nuevos. Este fenómeno se conoce como sobreajuste o *overfitting*. En la Figura 12 se puede ver el ejemplo de un modelo que presenta *overfitting*, así como las otras dos posibles situaciones que puede darse cuando el proceso de entrenamiento de un modelo de clasificación finaliza. En este sentido, el objetivo suele ser obtener el caso de *Appropirate-fitting*, para así obtener buenos resultados de clasificación en el *test set*.

Para resolver el *overfitting* existen diversas posibilidades: aumentar el tamaño de la base de datos para obtener un *training set* más variado y diverso, implementar algún método de regularización, implementar el método del *dropout* o implentar alguna combinación de las tres propuestas anteriores.

En muchos casos, el aumento de la base de datos no puede realizarse fácilmente, ya sea por conllevar un incremento de los costes, por la imposibilidad de obtener nuevas mediciones, lecturas o datos, etc. En estos casos, la única solución posible es recurrir a los métodos de regularización y *dropout* para intentar reducir el *overfitting* y mejorar la precisión del modelo proporcionado por la red neuronal en el *test set*.

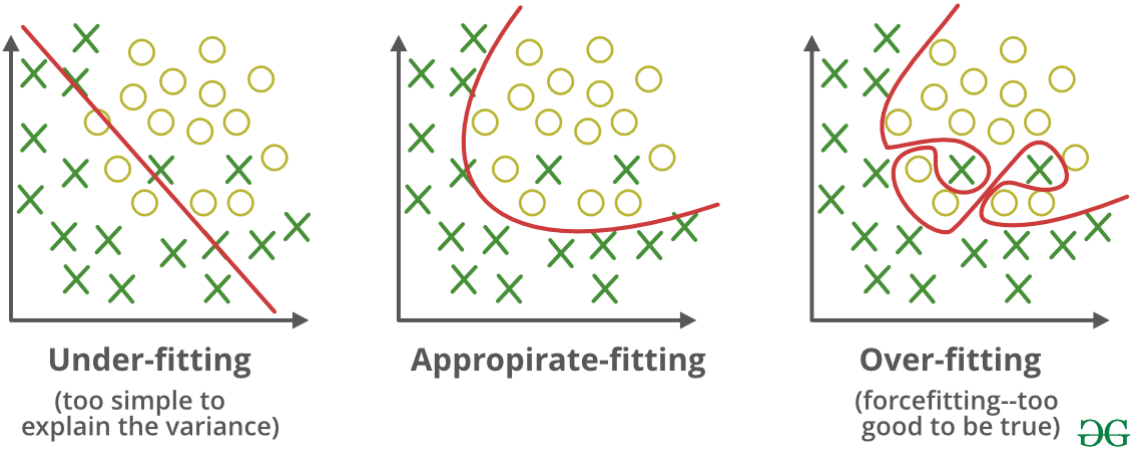
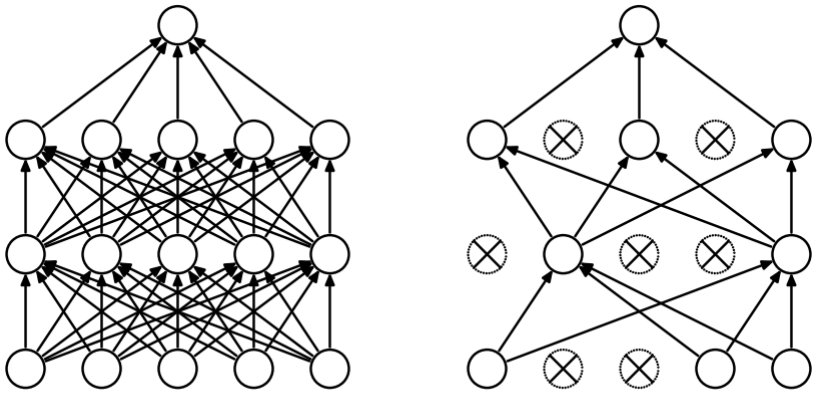


Figura 12: Representación gráfica de las tres situaciones que pueden darse cuando un modelo finaliza su entrenamiento [11].

De un lado, es probable que se produzca *overfitting* cuando el grado de libertad de la red neuronal es demasiado alto para los datos de entrenamiento. Sin embargo, como se ha comentado, en muchos casos los datos de entrenamiento no pueden aumentarse con facilidad y el grado de libertad de la red no puede reducirse fácilmente al estar profundamente involucrado en el poder expresivo de la representación [114]. Por lo tanto, se requiere un método de regularización para mitigar el *overfitting* proporcionando algún tipo de restricción en el parámetro de aprendizaje del sistema.

En particular, la regularización L2, también denominada disminución de peso, es eficaz para lograr un buen rendimiento de generalización en muchos casos [6]. Al utilizar la regularización L2, se añade un término a la función de pérdida (ver Ecuación 9) y a la función de actualización de los parámetros de la red (ver Ecuaciones 10 y 11) con el objetivo de penalizar la complejidad y el ajuste del modelo. Al minimizar el ajuste, se obtienen modelos más simples que tienden a generalizar mejor. En consecuencia, el modelo puede ser capaz de lograr una buena precisión en el *training set* y, al mismo tiempo, presentar una mejor precisión con datos nuevos del *test set*.

Por otro lado, el *dropout* es una técnica para evitar el *overfitting* ampliamente utilizada. El método funciona desactivando aleatoriamente las neuronas de una capa específica de la red de acuerdo a una tasa constante p en cada ejemplo del *training set* durante el proceso de entrenamiento. Es decir, en cada etapa del entrenamiento el aprendizaje se realiza como si ciertas unidades de la capa en la que se aplica *dropout*, y que son seleccionadas por el algoritmo, no existieran. Este método tiene como objetivo evitar que la red neuronal aprenda a depender de pesos específicos y, por tanto, se adapte demasiado a los datos del *training set* y no sea capaz de generalizar los resultados para el *test set*. En la Figura 13 se puede observar la configuración de una red neuronal durante el entrenamiento antes y después de aplicar *dropout*.



(a) Standard Neural Net (b) After applying dropout.

Figura 13: Esquema de una red neuronal antes y después de aplicar *dropout* [12].

3.3.2. Red Neuronal Recurrente

Una red neuronal recurrente es un tipo de red neuronal capaz de trabajar con secuencias temporales de datos. Este tipo de redes se utilizan ampliamente en tareas de reconocimiento de secuencias, como reconocimiento del habla [115], procesamiento de lenguaje [116], generación de música [117], clasificación de sentimientos [118], traducción [119] y reconocimiento de actividades o acciones [120, 121].

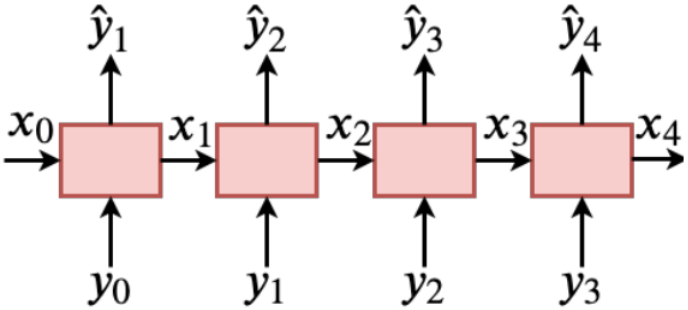


Figura 14: Representación de la estructura básica de una red neuronal recurrente [13].

En una capa de neuronas recurrentes, cada neurona recibe en cada instante de tiempo dos entradas: la entrada correspondiente de la capa anterior y la salida de la misma capa del instante anterior, como se puede ver en la Figura 14. De esta forma, cada neurona recurrente presenta dos conjuntos de parámetros, uno para la entrada de datos que recibe de la capa anterior y otro para la entrada de datos correspondiente a la salida del instante anterior. Esto se puede describir mediante la siguiente ecuación:

$$y_t = g(x_t W + y_{t-1} U + b) \tag{12}$$

donde W es la matriz de pesos y b el término de sesgo que actúan sobre la secuencia de entrada proveniente de la capa anterior x_t y U es la matriz de pesos que actúa sobre el estado de la red en

el instante de tiempo anterior y_{t-1}. Por lo tanto, en la fase de *backpropagation* del entrenamiento también se actualizarán los parámetros de la matriz U [101].

De esta forma, se puede afirmar que las neuronas recurrentes presentan cierta memoria, debido a que la salida de una neurona en un instante de tiempo determinado depende de una entrada que proviene de instantes de tiempo anteriores que contiene implícitamente información acerca de la historia de los elementos pasados de la secuencia [91].

La capacidad de tener memoria convierte a las redes recurrentes una herramienta adecuadas para tareas de *machine learning* que involucren datos secuenciales. Al poder utilizar información relevante proveniente de datos de entrada del pasado en el proceso de entrenamiento pueden realizar predicciones más precisas. Esta habilidad se plantea como un factor clave en tareas de clasificación de datos en las que la secuencia de datos y la dinámica temporal que conecta los datos a menudo es más importante que el contenido de cada dato individual. Además, marca la principal diferencia con respecto las redes neuronales estándares, al no disponer estas de la habilidad de almacenar información del pasado, a excepción de la información obtenida en el proceso de entrenamiento y reflejada en los parámetros de la red [101, 122].

Aunque las redes neuronales recurrentes son sistemas dinámicos muy potentes, entrenarlas puede provocar problemas debido a que durante la fase de *backpropagation* los gradientes aumentan o disminuyen en cada intervalo de tiempo. Si este crecimiento o disminución sucede durante muchos intervalos de tiempo, se puede producir un fenómeno conocido como explosión o desaparición de los gradientes que impida encontrar la adecuada configuración de los parámetros de la red [13].

Aunque el objetivo principal de las redes recurrentes es aprender las dependencias a largo plazo, almacenar información durante mucho tiempo no es una tarea sencilla. Para corregir esto se introducen las redes de memoria de largo y corto plazo (LSTM, del inglés *Long Short-Term Memory*), una extensión de las redes neuronales recurrentes que utilizan unidades ocultas especiales, cuyo objetivo es aumentar la memoria de la red para que sea capaz de recordar información importante a lo largo del tiempo [123].

Las unidades principales de las capas LSTM se denominan celdas de memoria. Una celda de memoria se compone de cuatro elementos principales: una puerta de entrada, una neurona con conexión autorecurrente, una puerta de olvido y una puerta de salida. En función de la importancia de la información de los datos de entrada proporcionados a la LSTM, las puertas de la celda de memoria pueden realizar diferentes operaciones: escribir (puerta de entrada), leer (puerta de salida) o borrar (puerta de olvido) información de la celda de memoria [124].

Por lo tanto, las puertas de las celdas de memoria determinan si se permite o no una nueva entrada, se elimina la información existente al considerar que no es importante o se deja que afecte a la salida en el instante actual de tiempo. La mayor o menor importancia de la información proporcionada por los datos se decide a través de los parámetros del sistema.

En resumen, las LSTM pueden capturar mejor las dependencias temporales de los datos de entrada a largo plazo, debido a la habilidad de las unidades de las LSTM para registrar posibles patrones de comportamiento temporal. Además, tienen la capacidad de evitar el fenómeno de explosión o desaparición de los gradientes [101].

No obstante, las redes recurrentes no tienen por qué utilizarse de forma independiente. Una

ventaja que presentan es su capacidad de ser modelos entrenables que pueden alimentar a otros componentes de la red para trabajar conjuntamente con ellos. Por ejemplo, la salida de una red recurrente puede alimentar a una red neuronal estándar que intentará predecir o clasificar un conjunto de datos [6, 125].

4. Resultados

Una vez estudiados los dos grupos de métodos de clasificación que se utilizaran en el proceso de clasificación de la relación social en parejas, es necesario aplicarlos a la base de datos para comprobar la precisión que ofrece cada uno de ellos y poder así seleccionar el método de clasificación más efectivo.

Inicialmente, en la sección 4.1 se detalla la cronología llevada a cabo a la hora de aplicar los métodos de clasificación. A continuación, se presentan los resultados numéricos obtenidos de dicha aplicación. De un lado, en la subsección 4.2 se detallan los resultados numéricos obtenidos al utilizar los diversos métodos de *clustering* explicados en la subsección 3.2. De otro lado, en la subsección 4.3 se presentan los resultados numéricos obtenidos al aplicar los métodos de *deep learning* estudiados en la subsección 3.3. Además, la subsección anterior incluye la subsección 4.3.1 que detalla los resultados de realizar una nueva clasificación de los ejemplos de la base de datos en dos nuevas categorías utilizando modelos de *deep learning*. Finalmente, en la subsección 4.4 se comparan los mejores resultados obtenidos en las subsecciones anteriores con los resultados del estudio [1].

4.1. Cronología

Como se ha comentado anteriormente, el objetivo principal de la tesis es clasificar los ejemplos de la base de datos según el tipo de relación existente entre los miembros de la pareja. Los posibles tipos de relación son colegas, pareja, familia o amistad.

Los primeros métodos utilizados para clasificar la base de datos son los métodos de *clustering*. Para los diferentes métodos de *clustering* se utilizan diferentes estrategias. De un lado, los métodos de *K-Means*, *hierarchical clustering* y *spectral clustering* se implementan definiendo el número de *clusters* deseado para la clasificación. En este caso, estos métodos se ejecutan para clasificar los datos en cuatro *clusters*, uno para cada tipo de relación posible.

Además, para el método de *hierarchical clustering* se utiliza la distancia euclidiana como medida de similitud y como criterio a optimizar se escoge la minimización de la varianza de los clusters fusionados. Para el método de *spectral clustering* se utiliza la función de base radial (RBF, del inglés *Radial Basis Function*), explicada en detalle en [126] y cuyo valor para dos muestras disminuye con la distancia, para construir la matriz de similitud.

Por otro lado, los métodos de *Mean-Shift* y *Affinity Propagation* se llevan a cabo sin definir un número de *clusters* concreto. La razón de no definir un número de *clusters* es utilizar estos dos métodos para comprobar cuantos *clusters* se forman para la base de datos y comprobar si el número de categorías de la clasificación se aproxima al obtenido al dar libertad a los métodos. Además, para el método de *Affinity Propagation* se utiliza la distancia euclidiana como medida de similitud.

Una vez obtenidos los resultados de los diferentes métodos de *clustering*, se recurre al *deep learning* y a la aplicación de redes neuronales. Inicialmente se empieza con una red neuronal estándar. A continuación, se utilizan diferentes diseños de redes neuronales para comprobar cual ofrece la mejor precisión, tanto en el *training set* como, especialmente, en el *test set*. En cada di-

seño implementado se varían diversos hiperparámetros de la red: el número de capas ocultas, el número de neuronas de las capas ocultas, el *learning rate* y el número de *epochs* del proceso de entrenamiento. Además, para intentar solucionar el problema del *overfitting* se utilizan los métodos de regularización L2 y *dropout*.

Para entrenar las diferentes redes neuronales diseñadas, la base de datos existente se divide en *training set* y *test set*. De esta forma, un $90\,\%$ de los datos se utilizan como *training set* para entrenar el modelo y el $10\,\%$ restante se utilizan para comprobar la precisión y validez del modelo entrenado como *test set*. Por lo tanto, si la base de datos está formada por 867 experimentos, un total de 780 ejemplos son los que conforman el *training set* y los restantes 87 ejemplos forman parte del *test set*.

Una vez implementados los modelos de red neuronal estándar, se procede a diseñar diversas redes neuronales recurrentes para comprobar la eficacia que presentan. Al tener lecturas de los ejemplos de la base datos a lo largo de un cierto período de tiempo, se decide implementar redes recurrentes debido a su capacidad para trabajar con secuencias temporales de datos y extraer posibles dependencias entre ellas.

Para entrenar las redes neuronales recurrentes es necesario adaptar la base de datos original. Para esto, se realiza un nuevo programa de extracción de datos que recorre los archivos de todos los experimentos. En cada experimento, selecciona las primeras 41 mediciones registradas. Como, generalmente, las lecturas se registran cada $0,5$ segundos, esto equivale a obtener las lecturas de los primeros 20 segundos de cada proceso de acompañamiento. Una vez obtenidas las mediciones de un experimento, se tiene una matriz de 41 (número de lecturas) por 16 (número de parámetros) y, a continuación, el programa repite este proceso para el siguiente experimento. Cuando se obtiene una nueva matriz de lecturas, se amplía en el eje Z la matriz anteriormente obtenida, creando así una matriz 3D que recoge las lecturas de los diferentes parámetros de todos los experimentos. Esta matriz se utiliza como entrada de la red neuronal recurrente.

Si un experimento no presenta un mínimo de 41 lecturas, es decir, que el proceso de acompañamiento dura menos de 20 segundos, no se utilizan los datos de ese experimento. De esta forma, el programa de extracción recoge los datos de un total de 771 ejemplos de los 867 experimentos de la base de datos inicial. Por lo tanto, la matriz de entrada de la red neuronal recurrente presenta unas dimensiones de $771x41x16$. La Tabla 2 muestra la distribución de los 771 ejemplos de parejas de la nueva base de datos en las cuatro categorías estudiada.

Categoría	**Nº de ejemplos**
Colegas	242
Pareja	85
Familia	181
Amistad	263
Total	771

Tabla 2: Distribución de los ejemplos de la nueva base de datos en las cuatro categorías estudiadas.

Siguiendo el mismo proceso que con las redes neuronales, los datos de entrada se dividen en *training set* y *test set* en una proporción de 90-10. Por lo tanto, los datos que forman parte del *training set* se recogen en una matriz de $693x41x16$, mientras que los datos del *test set* se encuentran en una matriz de $78x41x16$.

Existen diversas condiciones de diseño que se cumplen para todas las redes neuronales desarrolladas, tanto para las redes neuronales estándar como para las recurrentes. Estas condiciones son:

- La capa de entrada está formada por 16 neuronas. Cada neurona corresponde a uno de los parámetros de entrada explicados en la subsección 3.1.1.
- Las capas ocultas están diseñadas con la función de activación *ReLU*, explicada en la subsección 3.3.1.
- La capa de salida está diseñada con la función de activación *Softmax*, explicada en la subsección 3.3.1.
- Como algoritmo de *gradient descent* se utiliza el *Mini-Batch Gradient Descent*, explicado en la subsección 3.3.1.
- Como función de pérdida se utiliza la *croos-entropy function*, definida en la Ecuación 9.

Además, la primera capa oculta de las redes neuronales recurrentes es una capa LSTM, explicada en la subsección 3.3.2, que permite al modelo capturar las dependencias temporales de los datos de entrada a largo plazo.

Todos los métodos de clasificación implementados se programan en lenguaje Python. Además, las redes neuronales se diseñan utilizando TensorFlow, una plataforma de código abierto para el *machine learning* y la IA desarrollada por Google. Para los métodos de *clustering* y otras funciones, como por ejemplo la división de la base de datos en *training set* y *test set*, se utilizan funciones de la biblioteca de *machine learning* Scikit-learn[1] [5]. Los primeros modelos de redes neuronales estándar se diseñan utilizando TensorFlow en su versión 1 y adaptando los programas realizados durante el curso especializado en *deep learning* ofrecido en la web de Cousera[2]. Los siguientes modelos de red neuronal estándar y todas las redes neuronales recurrentes se realizan utilizando TensorFlow en su versión 2 y Keras, una biblioteca de redes neuronales de código abierto escrita en Python.

Finalmente, con el objetivo de analizar mejor los resultados obtenidos, se deciden fusionar las categorías de pareja, familia y amistad en una sola categoría llamada "íntimos", en referencia a la consideración que las tres relaciones fusionadas conllevan un grado de intimidad entre los miembros de la pareja superior al de la relación de colegas. En este sentido, la categoría de colegas se renombra como "conocidos" para llevar a cabo el proceso de clasificación. Por lo tanto, se modifica el programa de extracción de datos para clasificar los experimentos en las nuevas categorías definidas, y modificar así la base de datos de entrada de las redes neuronales.

Para evaluar los métodos de clasificación se utiliza un elemento conocido como matriz de confusión. Esta herramienta contabiliza las categorías reales de los ejemplos de la base de datos y las categorías asignadas a estos ejemplos por el algoritmo concreto implementado, permitiendo así observar y comparar la precisión de cada método a la hora de clasificar los datos en las diferentes categorías. La estructura de las matrices de confusión utilizadas se puede ver en la

[1] https://scikit-learn.org/stable/index.html
[2] https://www.coursera.org/specializations/deep-learning

Tabla 3. Tal y como se observa, los valores de la diagonal de la tabla corresponden a ejemplos clasificados correctamente, mientras que el resto hacen referencia a ejemplos incorrectamente clasificados por el método evaluado.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	Verdaderos Colegas	Falsas Parejas	Falsas Familias	Falsas Amistades
	Parejas	Falsos Colegas	Verdaderas Parejas	Falsas Familias	Falsas Amistades
	Familia	Falsos Colegas	Falsas Parejas	Verdaderas Familias	Falsas Amistades
	Amistad	Falsos Colegas	Falsas Parejas	Falsas Familias	Verdaderas Amistades

Tabla 3: Estructura de las matrices de confusión utilizadas para evaluar los métodos de clasificación.

A lo largo del desarrollo del proyecto surgieron diversos contratiempos que requirieron ser solucionados para seguir avanzando, de entre los que se destacan tres. El primero tuvo relación con la creación del programa base para la extracción de las lecturas de los ejemplos de la base de datos. Para poder obtener los datos era necesario recorrer todos los archivos, extraer los datos requeridos y volver a organizarnos en otro archivo con la estructura deseada. Este proceso resultó más complicado de lo esperado debido a la configuración de la base de datos proporcionada, pero, finalmente, se pudo obtener un programa que funcionaba de la manera esperada y se podía adaptar posteriormente a las necesidades de los métodos de clasificación.

El segundo contratiempo ocurrió durante el desarrollo de las redes neuronales estándar. Como se ha comentado anteriormente, inicialmente las redes neuronales fueron diseñadas utilizando TensorFlow en su versión 1, tal y como el autor de esta tesis había aprendido en el curso de *deep learning* que había realizado. No obstante, existen ciertas funciones de TensorFlow 1 que fueron eliminadas al actualizar TensorFlow a su versión 2 y, por lo tanto, fue necesario retocar el diseño inicial de redes neuronales. Además, para aprovechar al máximo el potencial de la nueva versión de TensorFlow se decidió aprender a diseñar redes neuronales en TensorFlow 2 y Keras de manera autodidacta.

Finalmente, el último problema vino derivado de la dificultad inicial de diseñar el programa que utiliza una red neuronal recurrente para realizar la clasificación. Concretamente, el principal contratiempo estuvo en adaptar las lecturas proporcionadas por el programa de extracción de datos a las características de la alimentación de la red. No obstante, todos los contratiempos pudieron ser superados y se lograron implementar los diversos métodos de clasificación con éxito.

4.2. Resultados numéricos *clustering*

En este apartado se presentan los resultados obtenidos utilizando los diferentes métodos de *clustering* introducidos en la subsección 3.2.

Al aplicar el método de *K-Means* explicado en la subsección 3.2.1, en el proceso de clasificación

se obtiene una precisión del 26 %. La representación de los datos y los *clusters* se observa en la Figura 15 y la matriz de confusión obtenida se recoge en la Tabla 4.

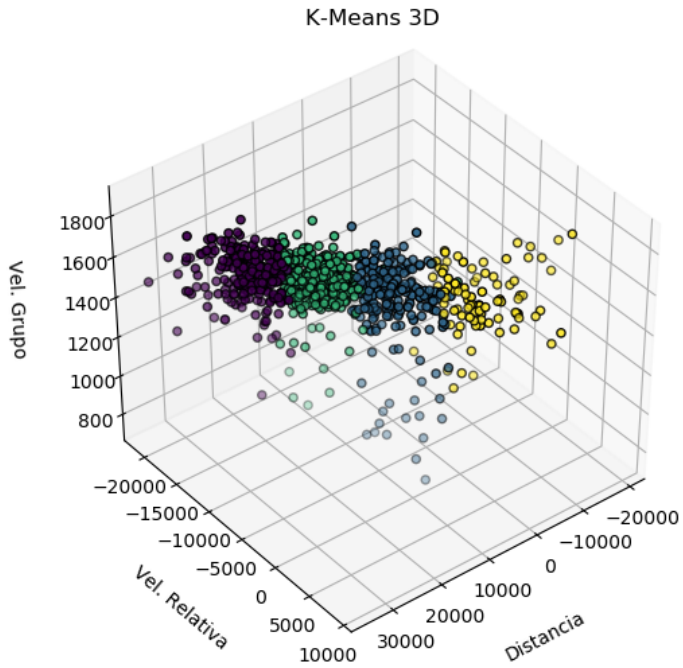


Figura 15: Representación en diferentes colores de los datos según el *cluster* al que pertenecen según el método *K-Means*.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	41,20	21,72	29,59	7,49
	Pareja	27,08	30,21	28,13	14,58
	Familia	21,56	33,03	31,19	14,22
	Amistad	33,22	23,78	34,97	8,04

Tabla 4: Matriz de confusión obtenida del método *K-Means* (en %).

En la Tabla 4 se destacan las precisiones del $41,20\,\%$ y $30,21\,\%$ obtenidas al clasificar correctamente los ejemplos de las categorías de colegas y pareja, respectivamente, al aplicar el método de *K-Means*.

Al aplicar el método de *agglomerative hierarchical clustering* explicado en la subsección 3.2.2, en el proceso de clasificación se obtiene una precisión del 30 %. La representación de los datos y los *clusters* se observa en la Figura 16. La matriz de confusión obtenida se recoge en la Tabla 5.

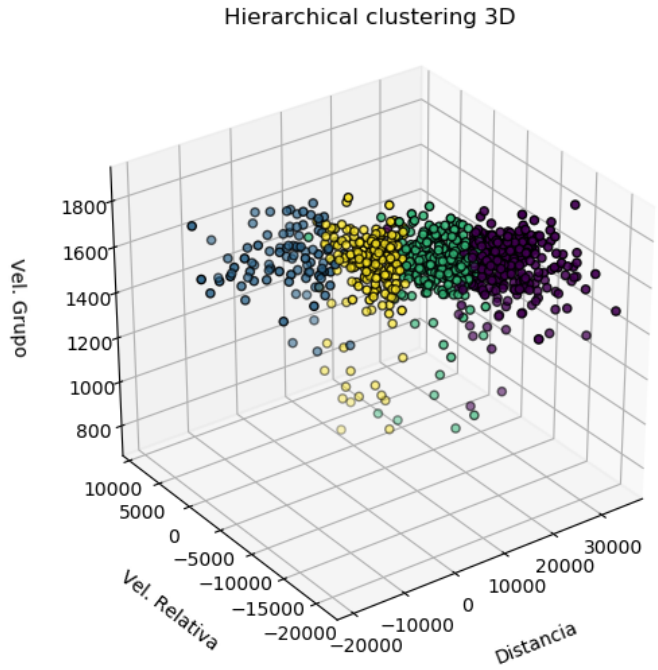


Figura 16: Representación en diferentes colores de los datos según el *cluster* al que pertenecen según el método *agglomerative hierarchical clustering*.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	46,44	8,61	26,97	17,98
	Pareja	32,29	14,58	26,04	27,08
	Familia	25,23	15,14	30,73	28,90
	Amistad	36,71	9,09	32,17	22,03

Tabla 5: Matriz de confusión obtenida del método *agglomerative hierarchical clustering* (en %).

En la Tabla 5 se destaca que los ejemplos de las categorías de colegas y familia son clasificados correctamente con unas precisiones del $46,44\,\%$ y $30,73\,\%$, respectivamente, al aplicar el método de *agglomerative hierarchical clustering*.

Al aplicar el método de *spectral clustering* explicado en la subsección 3.2.3, en el proceso de clasificación se obtiene una precisión del $31\,\%$. La representación de los datos y los *clusters* se observa en la Figura 17 y la matriz de confusión obtenida se recoge en la Tabla 6.

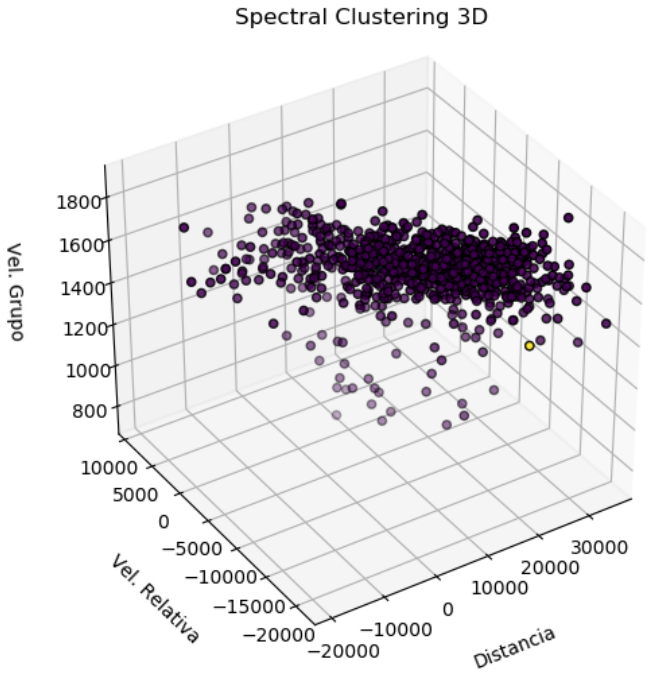


Figura 17: Representación en diferentes colores de los datos según el *cluster* al que pertenecen según el método *spectral clustering*.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	100	0	0	0
	Pareja	97,92	2,08	0	0
	Familia	98,17	0	0,92	0,92
	Amistad	100	0	0	0

Tabla 6: Matriz de confusión obtenida del método *spectral clustering* (en %).

En la Tabla 6 se destaca la buena precisión del modelo para clasificar los ejemplos de la categoría de colegas, pero las nefastas precisiones obtenidas para el resto de categorías al aplicar el método de *spectral clustering*.

Al aplicar el método de *Mean-Shift* explicado en la subsección 3.2.4, en el proceso de clasificación se estiman un total de 2 *clusters* para la base de datos de entrada. La representación de los datos y los *clusters* se observa en la Figura 18.

Al aplicar el método de *Affinity Propagation* introducido en la subsección 3.2.5, en el proceso de clasificación se estiman un total de 25 *clusters* para la base de datos de entrada. La representación de los datos y los *clusters* se puede ver en la Figura 19.

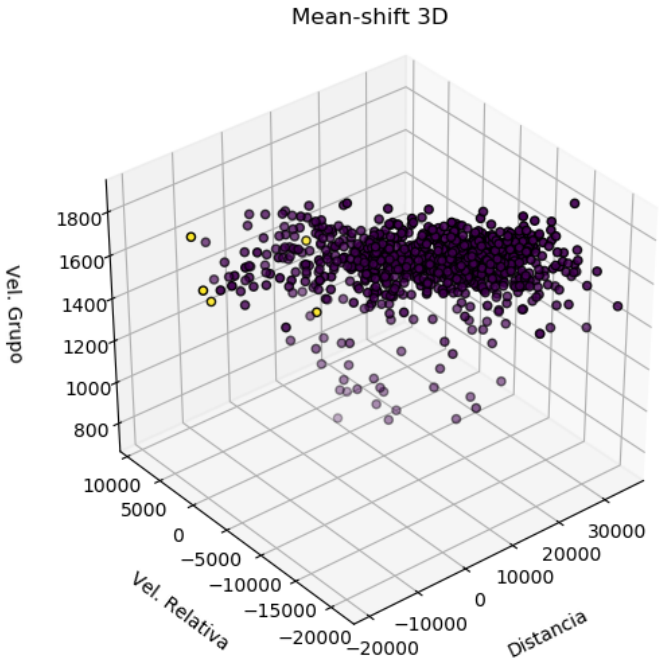


Figura 18: Representación en diferentes colores de los datos según el *cluster* al que pertenecen según el método *Mean-Shift*.

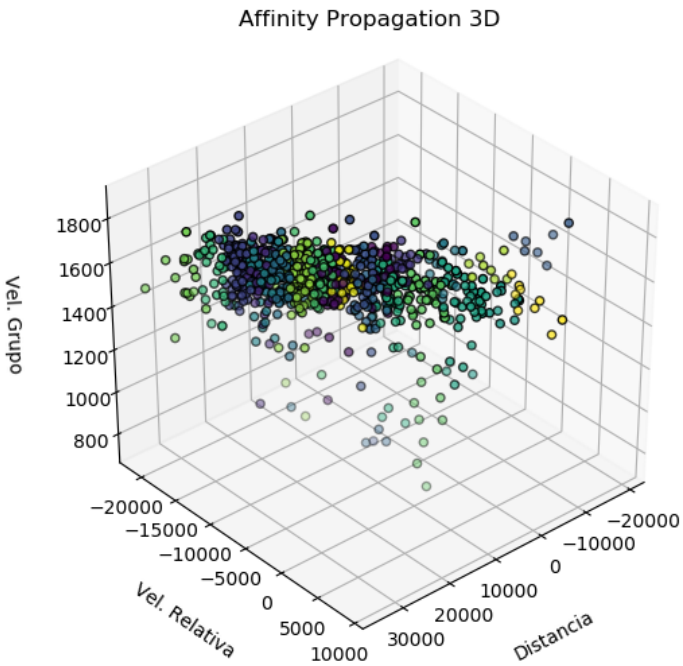


Figura 19: Representación en diferentes colores de los datos según el *cluster* al que pertenecen según el método *Affinity Propagation*.

Los resultados obtenidos por los métodos de *clustering* con el número de *clusters* definido previamente se recogen en la Tabla 7.

Método	Precisión
K-Means	26 %
Agglomerative hierarchical clustering	30 %
Spectral clustering	31 %

Tabla 7: Resumen de la precisión obtenida con los diferentes los métodos de *clustering* diseñados.

Al analizar los resultados expuestos en la Tabla 7, se observa que los métodos de *clustering* que requieren el número de *clusters* para realizar el análisis (*K-Means*, *agglomerative hierarchical clustering*, *spectral clustering*) presentan una precisión relativamente baja. De un lado, en la Tabla 6 se puede observar cómo, aunque la precisión para el método de *spectral clustering* es del 31 %, el algoritmo presenta una tendencia clara a clasificar los datos en la categoría de colegas, como se observa en la Figura 17. Por otro lado, aunque los métodos de *K-Means* y *agglomerative hierarchical clustering* presenten una precisión menor, al analizar las Tablas 4 y 5 y las Figuras 15 y 16 se observa cómo los algoritmos no presentan ningún sesgo hacía ninguna de las categorías.

Además, se observa que los métodos donde el número de *clusters* no se define previamente (*Mean-Shift* y *Affinity Propagation*) clasifican los datos en un número de *clusters* (2 y 25, respectivamente) que no coincide con el número de categorías originales, tal y como se comprueba en las Figuras 18 y 19. Esto puede ser debido a la poca diferencia que existe entre los parámetros que definen las relaciones entre personas y a la dificultad de clasificar las relaciones en las cuatro categorías estudiadas.

De un lado, la caracterización obtenida por el método *Mean-Shift* de 2 categorías puede deberse la creación de dos nuevas categorías que engloben a las cuatro categorías estudiadas y permitan ajustar mejor los parámetros de los ejemplos. Por otro lado, la clasificación en 25 categorías puede ser el resultado de la creación de subgrupos de diferente grado de familiaridad dentro de cada uno de los cuatro tipos de relaciones sociales estudiadas.

4.3. Resultados numéricos *deep learning*

En este apartado se presentan los resultados obtenidos utilizando los diferentes métodos de *deep learning* implementados en la subsección 3.3. Para empezar, se recogen los resultados de la clasificación obtenidos por las redes neuronales estándar explicadas en la subsección 3.3.1.

Las características de las redes neuronales estándar implementadas utilizando TensorFlow en la versión 1 se detallan en la Tabla 8. Los resultados obtenidos al ejecutar las redes definidas en la Tabla 8 se recogen en la Tabla 9. Los resultados de estos modelos se utilizan como primera aproximación al problema al estar diseñados en la versión 1 de TensorFlow, una versión actualmente desfasada. Por esta razón no se muestran las matrices de confusión de los modelos planteados. De esta forma, estos modelos sirven como punto de partida, en cuanto a la definición de los diferentes hiperparámetros, para los modelos realizados posteriormente.

	Nº de capas ocultas	Nº de neuronas de las capas ocultas	Nº de *epochs*	*Learning rate*
RN1-1	2	25-12	1500	0,00015
RN1-2	2	1500-600	1500	0,00015
RN1-3	2	1500-600	2500	0,00011
RN1-4	3	1800-1300-600	1500	0,00015
RN1-5	3	1500-1800-600	1500	0,00015

Tabla 8: Características de los modelos de red neuronal estándar implementadas en TensorFlow en la versión 1.

	Precisión *training set*	Precisión *test set*
RN1-1	54,36 %	41,38 %
RN1-2	92,82 %	35,63 %
RN1-3	98,97 %	33,33 %
RN1-4	38,20 %	34,48 %
RN1-5	58,84 %	35,63 %

Tabla 9: Precisión de los modelos de red neuronal estándar implementadas en TensorFlow en la versión 1.

A continuación, las características de las redes neuronales estándar implementadas utilizando TensorFlow en la versión 2 se detallan en la Tabla 10. Al ejecutar las redes definidas en la Tabla 10 se obtienen los resultados que se recogen en la Tabla 11. Para tratar de evitar el fenómeno del *overfitting*, explicado en la subsección 3.3.1, en algunos modelos se implementan métodos de regularización L2 y *dropout* (15 % en las capas ocultas). Las matrices de confusión de los diferentes métodos diseñados se recogen en diferentes tablas (12, 13, 14, 15, 16).

	Nº de capas ocultas	Nº de neuronas de las capas ocultas	Nº de *epochs*	*Learning rate*	Reg. L2	*Dropout*
RN2-1	2	25-12	1500	0,00015	No	No
RN2-2	2	1500-600	2500	0,00011	No	No
RN2-3	2	1500-600	2500	0,00011	Sí	No
RN2-4	4	1800-2500-1600-600	2500	0,00011	Sí	No
RN2-5	2	1500-600	2500	0,00011	Sí	Sí

Tabla 10: Características de los modelos de red neuronal estándar implementadas en TensorFlow en la versión 2.

	Precisión *training set*	**Precisión** *test set*
RN2-1	44,49 %	26,44 %
RN2-2	94,10 %	33,33 %
RN2-3	99,10 %	40,23 %
RN2-4	95,90 %	36,78 %
RN2-5	63,08 %	33,33 %

Tabla 11: Precisión de los modelos de red neuronal estándar implementadas en TensorFlow en la versión 2.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	4,17	45,83	0	50
	Pareja	7,14	50	0	42,86
	Familia	9,52	42,86	28,57	19,05
	Amistad	3,57	60,71	3,57	32,14

Tabla 12: Matriz de confusión del modelo RN2-1 (en %).

En la Tabla 12 se destacan las precisiones del $50\,\%$ y $32,14\,\%$ obtenidas al clasificar correctamente los ejemplos de las categorías de pareja y amistad, respectivamente, al utilizar el modelo RN2-1.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	33,33	4,17	25	37,50
	Pareja	21,43	14,29	21,43	42,86
	Familia	9,52	9,52	33,33	47,62
	Amistad	21,43	14,29	21,43	42,86

Tabla 13: Matriz de confusión del modelo RN2-2 (en %).

En la Tabla 13 se destaca que, al usar el modelo RN2-2, los ejemplos de las categorías de colegas y familia son clasificados correctamente con una precisión del $33,33\,\%$, mientras que los ejemplos de la categoría de amistad son clasificados correctamente con una precisión del $42,86\,\%$.

En la Tabla 14 se destaca que los ejemplos de las categorías de familia y amistad son clasificados correctamente con unas precisiones del $42,86\,\%$ y $50\,\%$, respectivamente, al utilizar el modelo RN2-3.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	37,50	4,17	29,17	29,17
	Pareja	28,57	21,43	28,57	21,43
	Familia	19,05	0	42,86	38,10
	Amistad	32,14	10,71	7,14	50

Tabla 14: Matriz de confusión del modelo RN2-3 (en %).

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	37,50	4,17	20,83	37,50
	Pareja	28,57	7,14	28,57	35,71
	Familia	14,29	9,52	47,62	28,57
	Amistad	21,43	14,29	21,43	42,86

Tabla 15: Matriz de confusión del modelo RN2-4 (en %).

En la Tabla 15 se destacan las precisiones del $47,62\,\%$ y $42,86\,\%$ obtenidas al clasificar correctamente los ejemplos de las categorías de familia y amistad, respectivamente, al usar el modelo RN2-4.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	33,33	0	20,83	45,83
	Pareja	21,43	0	35,71	42,86
	Familia	14,29	4,76	38,10	42,86
	Amistad	42,86	0	10,71	46,43

Tabla 16: Matriz de confusión del modelo RN2-5 (en %).

En la Tabla 16 se destaca que, al usar el modelo RN2-5, los ejemplos de las categorías de familia y amistad son clasificados correctamente con unas precisiones del $38,10\,\%$ y $46,43\,\%$, respectivamente.

A continuación, se recogen los resultados de la clasificación obtenidos por las redes neuronales recurrentes, explicadas en la subsección 3.3.2. En la Tabla 17 se detallan las características de las redes neuronales recurrentes implementadas utilizando TensorFlow en la versión 2. Los resultados obtenidos al ejecutar las redes definidas en la Tabla 17 se recogen en la Tabla 18. Para tratar de evitar el fenómeno del *overfitting*, explicado en la subsección 3.3.1, en todos los modelos se implementa el métodos de regularización L2. Las matrices de confusión de los diferentes métodos diseñados se recogen en diferentes tablas (19, 20, 21, 22, 23).

	Nº de capas ocultas	**Nº de neuronas de las capas ocultas**	**Nº de *epochs***	***Learning rate***	**Reg. L2**	***Dropout***
RNR2-1	2	25-12	1500	0,00015	Sí	No
RNR2-2	2	100-50	1000	0,00011	Sí	No
RNR2-3	2	1500-600	100	0,00011	Sí	No
RNR2-4	4	2500-1800-1200-600	10	0,00011	Sí	No
RNR2-5	2	1500-600	25	0,00011	Sí	No

Tabla 17: Características de los modelos de red neuronal recurrente implementadas en TensorFlow en la versión 2.

	Precisión *training set*	**Precisión *test set***
RNR2-1	45,17 %	41,03 %
RNR2-2	63,64 %	34,62 %
RNR2-3	67,97 %	38,46 %
RNR2-4	62,05 %	30,77 %
RNR2-5	66,52 %	42,31 %

Tabla 18: Precisión de los modelos de red neuronal recurrente implementadas en TensorFlow en la versión 2.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	82,61	0	8,70	8,70
	Pareja	33,33	0	0	66,67
	Familia	43,75	0	18,75	37,50
	Amistad	60,61	0	9,09	30,30

Tabla 19: Matriz de confusión del modelo RNR2-1 (en %).

En la Tabla 19 se destacan las precisiones del $82,61\,\%$ y $30,30\,\%$ obtenidas al clasificar correctamente los ejemplos de las categorías de colegas y amistad, respectivamente, al utilizar el modelo RNR2-1.

En la Tabla 20 se destaca que, al utilizar el modelo RNR2-2, los ejemplos de las categorías de colegas y amistad son clasificados correctamente con unas precisiones del $47,83\,\%$ y $36,36\,\%$, respectivamente.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	47,83	4,35	17,39	30,43
	Pareja	50	0	16,67	33,33
	Familia	43,75	0	25	31,25
	Amistad	36,36	9,09	18,18	36,36

Tabla 20: Matriz de confusión del modelo RNR2-2 (en %).

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	73,91	0	13,04	13,04
	Pareja	16,67	0	16,67	66,67
	Familia	56,25	0	43,75	0
	Amistad	60,61	3,03	18,18	18,18

Tabla 21: Matriz de confusión del modelo RNR2-3 (en %).

En la Tabla 21 se destaca que los ejemplos de las categorías de colegas y familia son clasificados correctamente con unas precisiones del $73,91\,\%$ y $43,75\,\%$, respectivamente, al utilizar el modelo RNR2-3.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	30,43	0	60,87	8,70
	Pareja	16,67	0	50	33,33
	Familia	18,75	0	62,50	18,75
	Amistad	30,30	0	48,48	21,21

Tabla 22: Matriz de confusión del modelo RNR2-4 (en %).

En la Tabla 22 se destacan las precisiones del $30,43\,\%$ y $62,50\,\%$ obtenidas al clasificar correctamente los ejemplos de las categorías de colegas y familia, respectivamente, al usar el modelo RNR2-4.

En la Tabla 23 se destaca que, al usar el modelo RNR2-5, los ejemplos de las categorías de colegas y familia son clasificados correctamente con unas precisiones del $69,57\,\%$ y $37,50\,\%$, respectivamente.

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	69,57	4,35	13,04	13,04
	Pareja	16,67	0	0	83,33
	Familia	50	0	37,50	12,50
	Amistad	48,48	3,03	15,15	33,33

Tabla 23: Matriz de confusión del modelo RNR2-5 (en %).

A primera vista, al analizar los resultados expuestos en las Tablas 9, 11, y 18 y compararlos con los resultados de la subsección 4.2, se observa que los métodos de *deep learning* presentan, en general, una precisión mayor que los métodos de *clustering*.

En la clasificación de la relación social en parejas en cuatro posibles categorías, se observa que las mejores precisiones en el *training set* se obtienen en los modelos basados en redes neuronales estándar y realizados con la versión 2 de TensorFlow, como se observa en la Tabla 11. Por el contrario, las mejores precisiones en el *test set* se obtienen en los modelos basados en redes neuronales recurrentes, como se observa en la Tabla 18. Concretamente, se destaca el modelo RN2-3, que presenta una precisión del $99,10\,\%$ en el *training set* y del $40,23\,\%$ en el *test set*, y el modelo RNR2-5, que presenta una precisión del $66,52\,\%$ en el *training set* y del $42,31\,\%$ en el *test set*.

Aunque las precisiones de los métodos en el *training set* y *test set* son unas métricas de evaluación básicas, es necesario realizar un análisis de las matrices de confusión obtenidas para comprender cómo se está realizando la clasificación de los ejemplos en las cuatro categorías estudiadas. Por ejemplo, si se tiene una base de datos formada por 90 ejemplos de la categoría A y sólo 10 ejemplos de la categoría B, un método de clasificación puede tener un sesgo hacia la categoría A. Si dicho método clasifica todos los ejemplos como clase A, su precisión será del $90\,\%$. Sin embargo, esto no significa que el método de clasificación funcione correctamente, ya que presenta un error del $100\,\%$ en la clasificación de los ejemplos de la categoría B.

De un lado, al analizar las matrices de confusión de los modelos basados en redes neuronales estándar que presentan una mejor precisión se observan diferentes fenómenos. En la Tabla 14, correspondiente al modelo que presenta una mayor precisión tanto en el *training set* como en el *test set*, se observa que el modelo RN2-3 reconoce las categorías correctas con una mayor precisión que las otras, llegando incluso al $50\,\%$ de precisión en la categoría de amistad, y a excepción de los ejemplos de la categoría de parejas, que son asignados en una proporción similar entre las cuatro categorías. Este mismo patrón se repite en la Tabla 15, correspondiente al modelo RN2-4, aunque empeorando la precisión de la clasificación de las categorías de amistad y pareja pero mejorando en la categoría de familia. El único modelo que presenta una precisión elevada en la categoría de pareja es el RN2-1 (ver Tabla 12), aunque presenta una eficacia relativamente baja al clasificar ejemplos en las otras categorías. En cuanto a los modelos RN2-2 y RN2-5 (ver Tablas 13 y 16, respectivamente), se observa que, en general, no son capaces de clasificar los ejemplos en las categorías correctas con una precisión superior a las otras y, además, la efectividad de la clasificación se ve superada en todos los casos por la del modelo RN2-3. De esta forma, el modelo RN2-3 se postula como el modelo basado una red neuronal estándar capaz de realizar una mejor clasificación de las relaciones humanas en las cuatro categorías estudiadas.

Por otra parte, al analizar las matrices de confusión generadas de los modelos basados en re-

des neuronales recurrentes se observan distintos fenómenos. Como se puede ver en la Tabla 19, correspondiente al modelo RNR2-1, se observa que el modelo presentado tiene preferencia al realizar la clasificación a asignar los ejemplos en las categorías de colegas o amistad, lo que resulta en una muy alta precisión para la categoría de colegas ($82,61\,\%$), pero nula para la categoría de familia. Esto último es un patrón que se repite para todos los modelos de redes recurrente, ya que todos los modelos diseñados presentan una precisión nula a la hora de clasificar correctamente los ejemplos de la base de datos en la categoría de pareja. Teniendo esto en consideración, el resto de modelos presenta una situación similar al tener más facilidades para clasificar los datos en ciertas categorías que en otras. Por ejemplo, mientras que los modelos RNR2-3 y RNR2-5 presentan una alta precisión al clasificar correctamente los ejemplos de la categoría de colegas (ver Tablas 21 y 23, respectivamente), el modelo RNR2-4 la tiene para la categoría de familia (ver Tabla 22). Además, los modelos basados en redes neuronales recurrentes presentan, en general, una considerablemente buena precisión al clasificar los ejemplos de la categoría de colegas en comparación con los modelos basados en redes neuronales estándar. Con todo esto, el modelo RNR2-5 parece plantearse como el modelo basado en una red neuronal recurrente que puede realizar una mejor clasificación de las relaciones humanas al presentar, en general, una eficacia relativamente mayor que los otros modelos para clasificar los ejemplos en las cuatro categorías correctas. No obstante, no se ha conseguido realizar un modelo capaz de obtener la mejor precisión en al menos tres de las categorías de relaciones, como si consigue lograr el modelo RN2-3.

Por lo tanto, se considera que el modelo que mejor puede realizar la clasificación de la relación social en parejas de humanos es el modelo RN2-3, al presentar la mejor precisión de todos los modelos implementados en el *training set*, además de las mejores precisiones en el *test set* y de las mejores eficacias en conjunto a la hora de realizar una correcta clasificación de los datos (ver Tabla 14).

Las diferencias entre las precisiones obtenidas en los modelos diseñados se deben a las distintas configuraciones de hiperparámetros que se han implementado en ellos. En función de estas configuraciones, las redes presentan unas precisiones en el *training set* y el *test set* mayores o menores. Aun así, al analizar las matrices de confusión de los modelos diseñados es posible extraer una serie de conclusiones.

Los modelos que presentan un número mayor de neuronas en sus capas ocultas o un mayor número de *epochs* son capaces de entrenar mejor a los parámetros de su red y, en consecuencia, obtienen precisiones elevadas en el *training set*. Sin embargo, la precisión disminuye ligeramente al aumentar el número de capas ocultas de dos a cuatro, como se observa al comparar los resultados de los modelos RN2-3 y RNR2-3 de dos capas ocultas con los resultados de los modelos RN2-4 y RNR2-4 de cuatro capas ocultas (ver Tablas 11 y 18). Este fenómeno puede producirse como resultado del aumento de la dificultad de aprendizaje durante el entrenamiento por el aumento del número de capas. Por lo tanto, se debe evitar un aumento innecesario del número de capas en el diseño del modelo, debido a que dicho aumento no tiene por qué conllevar un aumento de la precisión del modelo y, por el contrario, puede conducir a una reducción de la eficacia del mismo.

Otro factor a tener en cuenta es el tiempo de entrenamiento de los modelos. Cuando se aumenta el número de capas ocultas, el número de neuronas de las capas ocultas o el número de *epochs*, aumenta el tiempo de cálculo y el coste computacional del proceso de entrenamiento del modelo. En el caso concreto de este proyecto, se ha decidido dar prioridad a la precisión de los

modelos en la clasificación, pero en el caso de buscar un modelo con un período de entrenamiento más corto se deberá tener en cuenta la correcta elección de estos tres hiperparámetros para conseguir dicho objetivo.

Además, el método de regularización L2 consigue disminuir ligeramente el fenómeno del *overfitting* y aumentar la precisión del modelo, como se observa al comparar los resultados de los modelos RN2-2 y RN2-3 (ver Tabla 11). Por otro lado, el método de *dropout* no solo no consigue aumentar la precisión del modelo en el *test set*, sino que también disminuye considerablemente la precisión en el *training set*, como se observa al comparar los resultados de los modelos RN2-3 y RN2-5 (ver Tabla 11).

Por último, la elección del *learning rate* se plantea como un proceso en el que ir escogiendo diferentes valores e ir probando la eficacia de los modelos para cada uno de ellos hasta encontrar con el valor que ofrezca los mejores resultados posibles.

En consecuencia, el diseño de redes neuronales se plantea como un proceso iterativo de ensayo-error con el objetivo de encontrar la configuración de hiperparámetros que ofrezca la mejor eficacia del modelo y, por lo tanto, la mejor precisión posible en el proceso de clasificación de la base de datos. La Figura 20 representa este proceso iterativo en el que se parte de una idea, se programa el diseño correspondiente, se experimenta con el modelo para comprobar los resultados que ofrece y se vuelve al inicio del proceso.

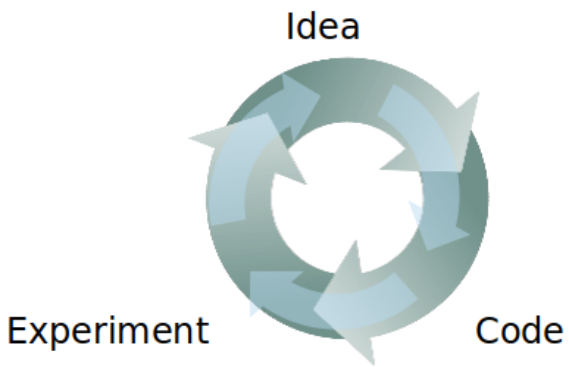


Figura 20: Representación del proceso iterativo de diseño de redes neuronales [14].

Por lo tanto, se puede afirmar que los métodos de *deep learning* basados en redes neuronales se posicionan como una herramienta relativamente eficaz a la hora de clasificar la relación entre personas de una pareja. No obstante, para que dichos métodos acaben de desplegar todo su potencial es necesario aumentar considerablemente la base de datos con más ejemplos de parejas, ya que es con grandes conjuntos de datos donde mejor trabajan las redes neuronales. Además, también se debe seguir con el proceso de configuración de hiperparámetros y probar nuevos modelos de redes neuronales para comprobar la eficacia que presentan.

Finalmente, se puede concluir que la diferenciación entre las cuatro categorías es un proceso complicado debido a la similitud que puede presentar datos de ejemplos de diversas categorías, incluso para un ser humano. Al no existir un manual de comportamiento para las diferentes relaciones sociales, encontrar un modelo capaz de identificar patrones entre diferentes ejemplos de la misma categoría puede suponer una tarea de gran dificultad. No obstante, los

mejores modelos presentados obtienen resultados relativamente buenos al realizar el proceso de clasificación de la relación social de las personas de una pareja en una de las cuatro categorías estudiadas. Para conseguir aumentar la eficacia del modelo es necesario aumentar los ejemplos de la base de datos. Particularmente, se tienen que aumentar los ejemplos cuya relación social es de pareja (ver Tabla 1) al ser, en general, la relación que presenta una peor precisión en la clasificación realizada por los modelos diseñados.

4.3.1. Clasificación en dos categorías

Debido a la dificultad para diferenciar entre las categorías de pareja, familia y amistad, se decide juntar estas tres categorías en una sola. Dicha categoría recibe el nombre de "íntimos", debido a que se considera que las tres relaciones conllevan un grado de intimidad entre los miembros de la pareja superior al de la relación de colegas restante. Además, la categoría de colegas se renombra como "conocidos".

A continuación, se recogen los resultados de la clasificación obtenidos por los dos tipos de redes neuronales utilizados. Las características de los diferentes modelos de redes neuronales implementadas utilizando TensorFlow en la versión 2 se detallan en la Tabla 24. Los resultados obtenidos al ejecutar las redes definidas en la Tabla 24 se recogen en la Tabla 25. Para tratar de evitar el fenómeno del *overfitting*, explicado en la subsección 3.3.1, en algunos modelos se implementan métodos de regularización L2 y *dropout* (15 % en las capas ocultas). Las matrices de confusión de los diferentes métodos diseñados se recogen en diferentes tablas (26, 27, 28, 29, 30).

	Nº de capas ocultas	**Nº de neuronas de las capas ocultas**	**Nº de *epochs***	***Learning rate***	**Reg. L2**	***Dropout***
RN2-6	2	25-12	700	0,00015	No	No
RN2-7	2	1500-600	2500	0,00011	No	No
RN2-8	2	1500-600	2500	0,00011	Sí	No
RN2-9	2	1500-600	2500	0,00011	Sí	Sí
RNR2-6	2	25-12	500	0,00011	Sí	No

Tabla 24: Características de los modelos de red neuronal implementadas en TensorFlow en la versión 2 para la clasificación binaria.

	Precisión *training set*	**Precisión *test set***
RN2-6	67,05 %	66,67 %
RN2-7	96,15 %	71,26 %
RN2-8	95,38 %	63,22 %
RN2-9	75,38 %	70,11 %
RNR2-6	75,76 %	65,38 %

Tabla 25: Precisión de los modelos de red neuronal implementadas en TensorFlow en la versión 2 para la clasificación binaria.

En la Tabla 26 se destaca la precisión del 82, 54 % obtenida al clasificar correctamente los ejemplos de la categoría de íntimos al utilizar el modelo RN2-6.

		Valor predicho	
		Conocidos	**Íntimos**
Valor real	**Conocidos**	25	75
	Íntimos	17,46	82,54

Tabla 26: Matriz de confusión del modelo RN2-6 (en %).

		Valor predicho	
		Conocidos	**Íntimos**
Valor real	**Conocidos**	45,83	54,17
	Íntimos	19,05	80,95

Tabla 27: Matriz de confusión del modelo RN2-7 (en %).

En la Tabla 27 se destaca que, al utilizar el modelo RN2-7, los ejemplos de las categorías de conocidos y íntimos son clasificados correctamente con unas precisiones del $45,83\,\%$ y $80,95\,\%$, respectivamente.

		Valor predicho	
		Conocidos	**Íntimos**
Valor real	**Conocidos**	33,33	66,67
	Íntimos	25,40	74,60

Tabla 28: Matriz de confusión del modelo RN2-8 (en %).

En la Tabla 28 se destaca que los ejemplos de la categoría de íntimos son clasificados correctamente con una precisión del $74,60\,\%$ al utilizar el modelo RN2-8.

En la Tabla 29 se destaca la buena precisión del modelo para clasificar los ejemplos de la categoría de íntimos, pero la nefasta precisión obtenida para la categoría de conocidos al usar el modelo RN2-9.

En la Tabla 30 se destaca que, al usar el modelo RNR2-6, los ejemplos de la categoría de íntimos son clasificados correctamente con una precisión del $85,45\,\%$.

		Valor predicho	
		Conocidos	**Íntimos**
Valor real	**Conocidos**	0	100
	Íntimos	3,17	96,83

Tabla 29: Matriz de confusión del modelo RN2-9 (en %).

		Valor predicho	
		Conocidos	**Íntimos**
Valor real	**Conocidos**	17,39	82,61
	Íntimos	14,55	85,45

Tabla 30: Matriz de confusión del modelo RNR2-6 (en %).

En la clasificación en dos posibles categorías se observa un aumento considerable de la precisión en el *test set* de los diferentes modelos diseñados, como se puede ver en la Tabla 25. Concretamente, se destaca el modelo RN2-7, que presenta una precisión del $96,15\,\%$ en el *training set* y del $71,26\,\%$ en el *test set*. Además, en la Tabla 27 se puede observar que el modelo presenta una elevada precisión en el reconocimiento de ejemplos de la categoría íntimos y la mejor precisión, con diferencia, de todos los modelos para la categoría conocidos. Por el contrario, otros modelos como el RN2-9 o el RNR2-6 presentan un sesgo muy considerable hacia la categoría de íntimos, como se observa en las Tablas 29 y 30. Por lo tanto, el modelo RN2-7 se postula como el modelo basado en una red neuronal capaz de realizar una mejor clasificación de las relaciones humanas en las dos categorías presentadas.

De esta forma, se puede afirmar que el proceso de clasificación de la relación social en dos categorías presenta una mayor precisión que la clasificación en cuatro categorías. Esto es debido a la fusión de las tres categorías que pueden presentar una mayor dificultad para distinguirse en una sola categoría. No obstante, la dificultad para diferenciar entre las categorías de pareja, familia y amistad y la categoría de colegas sigue suponiendo un problema en el proceso de clasificación. Como se ha comentado anteriormente, es necesario aumentar la base de datos de ejemplos y seguir con el proceso de configuración de hiperparámetros para conseguir un modelo que presente la mejor eficacia posible.

4.4. Comparación de resultados

Una vez obtenidos los resultados de los diferentes modelos de clasificación, estos se comparan con los resultados obtenidos en el estudio [1]. No obstante, es importante tener en cuenta que, aunque el objetivo de este proyecto coincida con el del estudio comparado, las bases de datos utilizadas no son exactamente iguales. Aun así, es posible realizar una comparación de los resultados obtenidos al utilizar métodos de clasificación diferentes.

Como en el estudio [1] se utilizan diversos métodos, parámetros e incluso datos, se utilizan los mejores resultados obtenidos por alguno de los métodos expuestos para compararlos con los resultados obtenidos por los modelos desarrollados en esta tesis. En concreto, se selecciona uno de los métodos del estudio que obtiene de las mejores precisiones de clasificación en las cuatro categorías de relaciones (ver Tabla 31).

		Valor predicho			
		Colegas	**Pareja**	**Familia**	**Amistad**
Valor real	**Colegas**	68,31	7,29	5,37	19,03
	Pareja	18,10	38,92	20,66	22,32
	Familia	13,58	31,11	36,57	18,75
	Amistad	34,19	16,66	12,74	36,41

Tabla 31: Matriz de confusión de uno de los métodos implementados en [1] (en %).

Por lo tanto, se comparan los resultados de uno de los mejores modelos del estudio [1], recogidos en la Tabla 31, con los resultados del modelo RN2-3 desarrollado en esta tesis (ver Tabla 14), considerado el modelo diseñado que mejor puede realizar la clasificación de la relación social en parejas de humanos. De esta forma, se observa que mientras el método del estudio presenta una mejor precisión en las categorías de colegas y parejas, el modelo desarrollado tiene una precisión mayor en las categorías de familia y amistad. No obstante, el método implementando en [1] presenta unas mayores diferencias en las precisiones de la clasificación de las categorías de colegas y parejas con respecto al método implementado en este proyecto, que las que presenta este método en las categorías de familia y amistad con respecto a las del estudio comparado. En consecuencia, es difícil afirmar con certeza qué modelo realiza una mejor clasificación, en general, de las relaciones sociales en las cuatro categorías presentadas.

Aun así, el potencial que ofrecen los modelos de *deep learning* permite sugerir que un aumento de la base de datos de ejemplos de parejas acompañándose y unos pequeños ajustes en la configuración de los hiperparámetros del modelo diseñado podrían suponer la mejora necesaria para obtener una mayor eficacia en el proceso de clasificación.

5. Planificación temporal

Uno de los aspectos más importante en cualquier proyecto de ingeniería es la planificación temporal de este. Una correcta planificación tiene en cuenta una distribución adecuada de las tareas del proyecto a lo largo del tiempo y permite adaptarse a posibles contratiempos que pueden surgir.

De un lado, en la subsección 5.1 se detalla la planificación planteada al inicio del proyecto de esta tesis relativa a la realización de las diferentes tareas del proyecto. De otro lado, en la subsección 5.2 se presenta la planificación final llevada a cabo y la distribución temporal de las distintas etapas del proyecto.

5.1. Planificación esperada del proyecto

En esta sección se presenta el plan de trabajo planificado inicialmente (sin considerar la posterior ampliación del Trabajo de Fin de Máster), el cual consta de siete tareas. Estas tareas se realizan para cumplir con los objetivos presentados en la sección 1.2. A continuación, se presenta una breve descripción de cada una de las tareas a realizar. Estas descripciones pretenden ser solo una orientación del trabajo requerido para cumplir con los objetivos del proyecto. En la Figura 21 se incluye un diagrama de Gantt con la distribución temporal planificada para cada tarea.

- **Tarea 1 - Redacción de la propuesta y del plan de trabajo**

  Esta primera tarea consiste en la redacción de la propuesta y plan de trabajo a realizar durante el proyecto de máster.

- **Tarea 2 - Estudio del estado del arte y los métodos a implementar**

  Esta tarea consiste en el estudio del estado del arte y de los métodos de clasificación a implementar.

  Se leerá el estado del arte proporcionado por tutor y ponente. Se buscarán y leerán otros artículos y estudios dentro de los subgrupos de estado del arte propuestos. Los subgrupos del arte propuestos son: (1) *proxemics* de Hall entre personas, y también entre robots y personas, (2) distancias sociales, (3) clasificación de comportamientos de personas caminando en grupos, (4) interacciones entre personas y robots y (5) navegación y acompañamiento social de robots respecto a personas.

- **Tarea 3 - Implementación del código en Python**

  Esta tarea consiste en la implementación de los métodos de teoría en Python. En esta tarea también se pueden usar códigos o funciones de distintas bibliotecas de código abierto encontradas por internet para ser utilizados como base o funciones auxiliares y posteriormente mejorados durante la implementación. Esta tarea comprende las subtareas siguientes:

  Sub-Tarea 3.1 - Buscar, entender y usar una base de datos ya existente que incluya relaciones entre grupos de personas, ya sea en datos obtenidos con láser o imágenes de

vídeo

Para poder realizar el proyecto es necesaria una base de datos que contenga imágenes o lecturas de grupos de personas navegando y la clasificación de la relación entre las personas de dichos grupos. Una vez localizada la base de datos, es indispensable comprender sus características para poder trabajar sobre ella o poder adaptarla a las necesidades de los métodos de clasificación a implementar.

Sub-Tarea 3.2 - Crear el clasificador de relaciones durante el acompañamiento

El objetivo de esta tarea es crear un clasificador que sea capaz de identificar la relación entre parejas de humanos cuando se están acompañando. Para esto se investigarán diferentes alternativas de métodos de clasificación para implementar en Python con el objetivo de procesar la base de datos obtenida y obtener una clasificación precisa de los ejemplos de la base de datos.

- **Tarea 4 - Extracción de resultados**

La tarea se centra en la extracción de resultados mediante el clasificador desarrollado y usando la base de datos proporcionada y/u otro tipo de vídeos de grupos de personas andando dentro de entornos urbanos. Según el método usado, esta tarea puede contener las subtareas siguientes:

Sub-Tarea 4.1 - Estudiar la base de datos proporcionada y dividirla en datos de entrenamiento y de prueba

Estudiar la base de datos proporcionada y dividirla en datos de entrenamiento (*training set*) y de prueba (*test set*), con el objetivo de poder entrenar el clasificador utilizando los datos del *training set* y, posteriormente, poder ponerlo a prueba con los datos del *test set*.

Sub-Tarea 4.2 - Definir/encontrar las características de las diferentes clases de acompañamiento entre personas

Pensar, analizar y comprender qué posibles tipos/clases de acompañamiento se pueden dar entre las personas.

Sub-Tarea 4.3 - Entrenar el clasificador para diferenciar las diferentes clases

En esta tarea se entrenará el clasificador y/o clasificadores implementados, usando los datos de la base de datos destinados al entrenamiento del clasificador (*training set*).

Sub-Tarea 4.4 - Probar el clasificador con nuevos datos

El objetivo es demostrar el correcto funcionamiento del clasificador, mediante la correcta clasificación de relaciones durante el acompañamiento incluidas en nuevos datos (*test set*) con los que el clasificador no ha sido entrenado.

Sub-Tarea 4.5 - Análisis de los resultados obtenidos en los clasificadores implementados

En esta tarea se evaluarán los resultados obtenidos para demostrar que el sistema funciona

en simulación. Además, se extraerán conclusiones de la investigación realizada.

Sub-Tarea 4.6 - Comparación de los resultados obtenidos en los clasificadores implementados con los resultados obtenidos por Francesco Zanlungo

En esta tarea se compararán los resultados obtenidos por los diferentes métodos de clasificación implementados, o por el método implementado que presente mejores resultados, con los resultados obtenidos por Francesco Zanlungo en el artículo [1].

- **Tarea 5 - Redacción del proyecto de final de máster**

  Esta tarea considera el tiempo mínimo de redacción del documento de la tesis por parte del estudiante. No obstante, sería aconsejable empezar a redactar la tesis desde que empieza la clasificación de datos por parte del clasificador. De esta forma, mientras este clasifica, ya se podría empezar con la redacción del documento de la tesis a ratos.

- **Tarea 6 - Preparación de la presentación para la defensa TFM**

  Preparar la presentación y la explicación de la defensa del TFM.

- **Tarea 7 - Defensa TFM**

  Realizar la defensa del TFM entre el 5 y el 14 de octubre.

- **Tarea 8 - Redacción del artículo**

  Redactar un artículo en inglés (*paper*) para ser posteriormente enviado a un congreso, por ejemplo, ICRA, IROS o HRI.

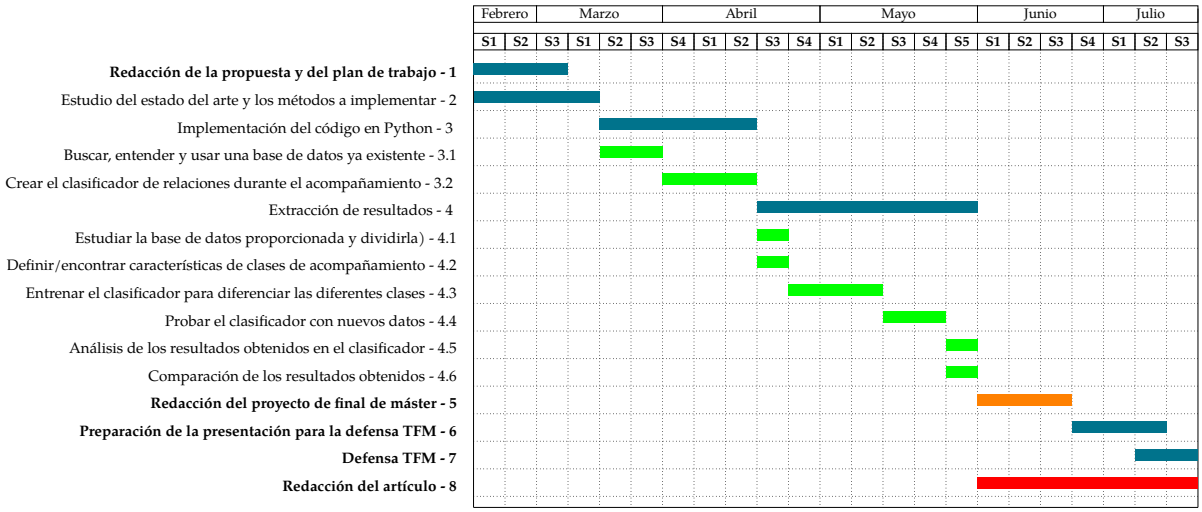


Figura 21: Plan de trabajo previsto del proyecto de final máster.

5.2. Planificación final del proyecto

A causa del volumen de trabajo encontrado durante la realización del proyecto fue necesaria realizar una ampliación del período para realizar la tesis de máster. De esta forma, la distribución de tareas en el tiempo varió respecto a la planificación inicial.

Además, la tarea 2 cuyo objetivo era estudiar acerca del estado del arte y de los métodos de clasificación a implementar, se dividió finalmente en dos subtareas:

- **Tarea 2 - Estudio del estado del arte y los métodos a implementar**

  Esta tarea consiste en el estudio del estado del arte y de los métodos de clasificación a implementar.

  Sub-Tarea 2.1 - Lectura del estado del arte y los métodos a implementar

  Se leerá el estado del arte proporcionado por tutor y ponente. Se buscarán y leerán otros artículos y estudios dentro de los subgrupos de estado del arte propuestos. Los subgrupos del arte propuestos son: (1) *proxemics* de Hall entre personas, y también entre robots y personas, (2) distancias sociales, (3) clasificación de comportamientos de personas caminando en grupos, (4) interacciones entre personas y robots y (5) navegación y acompañamiento social de robots respecto a personas.

  Sub-Tarea 2.2 - Realización de un curso especializado en *deep learning* y redes neuronales

  Esta tarea consiste en la realización de un curso especializado en *deep learning* y redes neuronales para profundizar en el campo del *deep learning*, aprender a diseñar diversos tipos de redes neuronales y averiguar su posible potencial como método de clasificación.

En la figura 22 se presenta diagrama de Gantt con la distribución temporal final realizada para cada tarea.

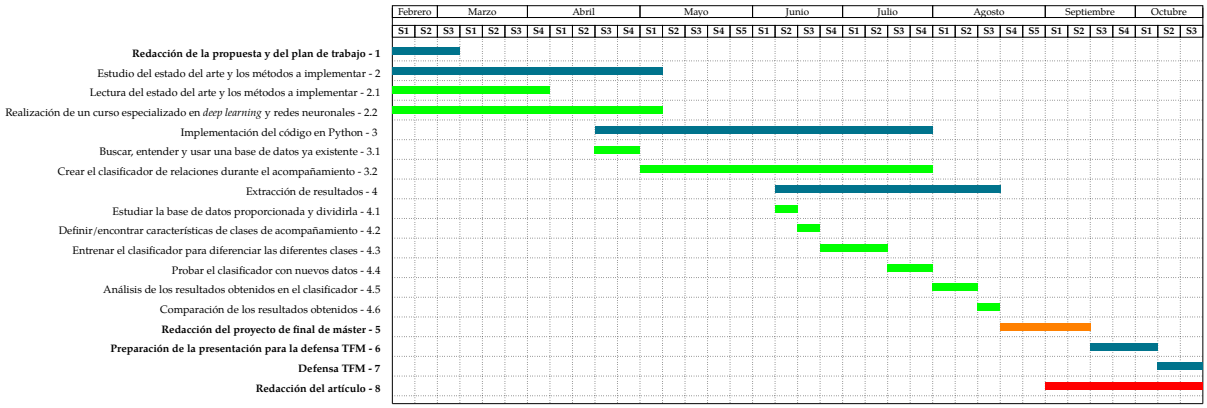


Figura 22: Plan de trabajo final del proyecto de final máster.

6. Estudio económico

El estudio económico del proyecto permite evaluar de forma cuantitativa el trabajo realizado y valorar si la realización del proyecto es viable desde el punto de vista económico. En este sentido, en esta sección se detalla el coste de la realización del proyecto de la presente tesis. Los costes del proyecto se dividen en dos apartados:

- Coste de los recursos humanos
- Coste del equipo y formación

En esta sección, el coste de los recursos humanos se detalla en la subsección 6.1, mientras que el coste del equipo y formación se detalla en la subsección 6.2. Finalmente, el coste total del proyecto se recoge en la subsección 6.3.

6.1. Coste de los recursos humanos

El coste de los recursos humanos hace referencia al coste (euros/hora) que conlleva disponer de un ingeniero titulado realizando el proyecto. Para poder calcular el coste del proyecte, las tareas realizadas a lo largo del proyecto, explicadas en la subsección 5.2, se engloban en cuatro grandes grupos:

- Coste asociado al estudio del estado del arte
- Coste asociado al diseño de los métodos de clasificación
- Coste asociado al análisis de resultados
- Coste asociado a la redacción del proyecto

Considerando un coste por hora de 30 €, el coste total de los recursos humanos se recoge en la Tabla 32.

Concepto	Horas dedicadas	Coste
Estudio del estado del arte	80	2.400 €
Diseño de los métodos	135	4.050 €
Análisis de resultados	15	450 €
Redacción del proyecto	70	2.100 €
Total	**300**	**9.000 €**

Tabla 32: Desglose del coste de los recursos humanos.

6.2. Coste del equipo y formación

El coste del equipo y formación hace referencia a los costes relacionados con el material y equipamientos necesarios para llevar a cabo el proyecto y al coste del curso realizado para formarse en *deep learning*. De esta forma, el coste total del equipo y formación se recoge en la Tabla 33.

Concepto	Coste
Formación en *deep learning*	160 €
Amortización equipo informático	52 €
Total	**212 €**

Tabla 33: Desglose del coste del equipo y formación.

6.3. Coste total del proyecto

Finalmente, en la Tabla 34 se muestra el coste total de la realización del proyecto recogido en esta tesis.

Concepto	Coste
Coste de los recursos humanos	9.000 €
Coste del equipo y formación	212 €
Total (sin IVA)	**9.212 €**
Total (con 21 % IVA)	**11.146,52 €**

Tabla 34: Coste total del proyecto.

7. Impacto del proyecto

Antes, durante y después de la realización de cualquier proyecto es de vital importancia valorar el impacto que el proyecto ha producido en el pasado, está produciendo en el presente o puede producir en el futuro en las distintas realidades que rodean al proyecto, como por ejemplo en la naturaleza o en las personas y sus sociedades.

En el caso del proyecto de esta tesis, se considera que el proyecto puede tener impacto únicamente en el ámbito social que afecta a las personas. Como el proyecto ha sido desarrollado utilizando solamente el equipo informático personal del autor, se considera que su impacto ambiental es mínimo y no discernible del consumo personal habitual.

No obstante, si en futuros proyectos se mejoran los métodos diseñados en esta tesis para que puedan ser implementados en un robot real, sí que podría producirse un posible impacto ambiental por parte del robot. Al navegar en entornos públicos poblados por humanos el robot podría afectar de manera negativa a su zona más próxima, por ejemplo, causando contaminación acústica por el ruido de sus motores o posibles sistemas de comunicación instalados en él.

7.1. Impacto social

Con el paso de los años se puede observar cómo cada día existen más sistemas con la capacidad de conocer los gustos y preferencias de las personas, de seguir y controlar a individuos en entornos públicos o de guardar imágenes o conversaciones personales durante años. Los avances en tecnología deben ir acompañados de un debate ético sobre la correcta utilización de estos y sobre su efecto en la vida y derechos de las personas.

En este sentido, la realización de un método de permita identificar la relación existente entre dos personas podría suponer para algunas personas una vulneración de su derecho a la privacidad. Por lo tanto, es importante tener en cuenta dos aspectos en lo respectivo a la recogida de datos:

- Es indispensable tratar los ejemplos de la base de datos y las nuevas lecturas de forma anónima.
- Si la recogida de datos se realiza mediante imágenes y estas van a ser mostradas de manera pública, es necesaria la autorización expresa de las personas que en ellas aparezcan o la posterior despersonalización de dichos datos (por ejemplo, haciendo borrosas y no identificables todas las caras de las personas que puedan aparecer en los citados vídeos).

En resumen, en un mundo donde la IA está cada día más presente debido a las infinitas posibilidades que ofrece es importante no olvidar que toda la tecnología desarrollada debe respetar los derechos de las personas a las que afecte, concretamente en el caso de este proyecto, su derecho a la privacidad.

Por otro lado, también es importante tener en cuenta la ventaja que supone el hecho que un sistema robótico se pueda adaptar de la forma más confortable posible a una persona mientras la acompaña a un destino. Lo cual puede ser muy beneficioso, por ejemplo, a la hora de usar robots asistentes para personas con necesidades especiales, enfermos o ancianos en residencias. Dichos

robots también pueden dar soporte a los trabajadores de centros de asistencia, colaborando y ayudando a disminuir la gran carga de trabajo que desempeñan. En definitiva, proporcionar a los robots las herramientas que les permitan adaptar su comportamiento de forma personalizada y comportarse de la manera más social posible solo puede traer beneficios para la sociedad del futuro.

Conclusiones

El proyecto realizado a lo largo de la presente tesis tiene como objetivo principal desarrollar un sistema capaz de clasificar el comportamiento entre dos personas en base al tipo de relación social existente entre ellas (colegas, pareja, familia o amistad) mientras se dirigen a un mismo objetivo. Para ello, ha sido necesario llevar a cabo un estudio del estado del arte actual, incluyendo tanto las ciencias sociales como el mundo de la robótica.

Una vez establecido el contexto, se ha investigado sobre distintos métodos de clasificación de datos que cuadraran con las características y necesidades de la tarea a desarrollar y se ha realizado la búsqueda de una base de datos sobre la que trabajar [1]. A continuación, se han implementado varios métodos de clasificación de relaciones sociales durante el acompañamiento entre personas. Concretamente, se han diseñado diversos métodos de *clustering* y se han desarrollado varios modelos de *deep learning* utilizando redes neuronales.

Después de implementar los métodos de clasificación, se han analizado los resultados obtenidos por los distintos métodos a partir de las precisiones que presentaban en el *training set*, en el *test set* y, gracias a las matrices de confusión, en la clasificación de los datos del *test set* en las cuatro categorías estudiadas.

Del análisis de resultados se puede concluir que los métodos de *K-Means* y *agglomerative hierarchical clustering* presentan unos resultados mediocres y su eficacia y capacidad de mejora son superadas por los modelos basados en redes neuronales. Concretamente, se destaca el modelo RN2-3 basado en una red neuronal estándar con el cual se obtienen unas precisiones relativamente buenas en el proceso de clasificación de la relación social entre dos personas.

Finalmente, se ha realizado el estudio económico del proyecto y se ha evaluado su posible impacto ambiental y social. Por lo tanto, en base a los objetivos establecidos al inicio del proyecto se puede afirmar que estos se han cumplido satisfactoriamente.

Cuando una persona se mueve por un entorno urbano y ve una pareja de peatones, no siempre resulta evidente adivinar la relación social existente entre las dos personas. Por el contrario, este proceso puede resultar encarecidamente complicado debido a las diferencias mínimas que pueden existir entre diferentes categorías de relaciones.

De los resultados obtenidos se puede concluir que el diseño de redes neuronales es una tarea compleja que requiere el ajuste de una gran variedad de parámetros. La modificación de algún factor o un cambio en el diseño puede conllevar una ejecución diferente y, en consecuencia, unos resultados muy distintos. En ese sentido, es necesario un extenso trabajo de prueba-error hasta encontrar la configuración de parámetros que mejor se ajuste a las necesidades buscadas y a la base de datos tratada con el objetivo de conseguir el mejor desempeño posible.

Los avances científicos y tecnológicos en campos como la informática o el *big data* permiten a los científicos e ingenieros reunir datos e interpretarlos para lograr una mejor comprensión del comportamiento humano y crear programas que consigan mejorar la adaptación de los robots sociales a nuestro día a día y allanar el camino a su completa integración en las sociedades humanas.

Los siguientes pasos a dar en el futuro estarían orientados a dos objetivos distintos. De un la-

do, el primer objetivo consistiría en intentar mejorar la precisión de las redes neuronales. Para conseguirlo es indispensable aumentar la base de datos con más ejemplos etiquetados de parejas navegando y, entonces, seguir ajustando los parámetros de los modelos y modificando su diseño para aumentar su efectividad.

Por otro lado, el segundo objetivo a llevar a cabo sería la implantación de los modelos diseñados en un robot real con los instrumentos necesarios para recoger datos de las parejas objetivo. Si esta fase se ejecutara con éxito, el siguiente paso a realizar consistiría en adaptar los programas diseñados para considerar al robot como uno de los miembros de la pareja. Entonces, el robot debería obtener los datos de diversos parámetros de la navegación del acompañante humano mientras este se desplaza para poder procesarlos a través de los nuevos modelos implementados y clasificar así su comportamiento en una de las cuatro categorías estudiadas. Finalmente, el robot tendría la información necesaria para poder adaptarse al comportamiento mostrado por su acompañante humano.

Agradecimientos

A mi padre y a mi madre, gracias por confiar siempre en mí y dejarme recorrer mi propio camino.

A mi abuelo, gracias por hacerme sentir la persona más importante del mundo.

A mi abuela, gracias por enseñarme lo que significa el servicio incondicional a los demás y la fuerza de voluntad.

A mis joves, gracias por hacerme crecer y mejorar cada día.

A mí mismo, gracias por aguantar hasta el final. Ha sido duro, pero también divertido.

Bibliografía

[1] Zeynep Yucel, Francesco Zanlungo, Claudio Feliciani, Adrien Gregorj, and Takayuki Kanda. Identification of social relation within pedestrian dyads. *PloS one*, 14(10):e0223656, 2019.

[2] Wikimedia Commons. Personal space, 2009.

[3] Ely Repiso, Gonzalo Ferrer, and Alberto Sanfeliu. On-line adaptive side-by-side human robot companion in dynamic urban environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 872–877. IEEE, 2017.

[4] Jacob Ávila. Clustering analysis, 2018.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis lectures on human language technologies*, 10(1):1–309, 2017.

[7] Ichi.Pro. Funciones de activación: conceptos básicos de sigmoid, relu, leaky relu y softmax para redes neuronales y aprendizaje profundo.

[8] Ji Yang. Relu and softmax activation functions, 2017.

[9] Divakar Kapil. Stochastic vs batch gradient descent, 2019.

[10] Fernando Berzal. Entrenamiento de redes neuronales, 2018.

[11] GeeksforGeeks. Ml underfitting and overfitting, 2021.

[12] 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.

[13] Antônio H Ribeiro, Koen Tiels, Luis A Aguirre, and Thomas Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In *International Conference on Artificial Intelligence and Statistics*, pages 2370–2380. PMLR, 2020.

[14] Andrew Ng. Redes neurales y aprendizaje profundo, 2017.

[15] H-M Gross, H Boehme, Ch Schroeter, Steffen Müller, Alexander König, Erik Einhorn, Ch Martin, Matthias Merten, and Andreas Bley. Toomas: interactive shopping guide robots in everyday use-final implementation and experiences from long-term field trials. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2005–2012. IEEE, 2009.

[16] Noriaki Hirose, Ryosuke Tajima, and Kazutoshi Sukigara. Personal robot assisting trans-

portation to support active human life. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5395–5402. IEEE, 2015.

[17] Felix Faber, Maren Bennewitz, Clemens Eppner, Attila Gorog, Christoph Gonsior, Dominik Joho, Michael Schreiber, and Sven Behnke. The humanoid museum tour guide robotinho. In *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 891–896. IEEE, 2009.

[18] Andrea Bauer, Dirk Wollherr, and Martin Buss. Human–robot collaboration: a survey. *International Journal of Humanoid Robotics*, 5(01):47–66, 2008.

[19] Iolanda Leite, Carlos Martinho, and Ana Paiva. Social robots for long-term interaction: a survey. *International Journal of Social Robotics*, 5(2):291–308, 2013.

[20] Yoichi Morales, Naoki Akai, and Hiroshi Murase. Personal mobility vehicle autonomous navigation through pedestrian flow: A data driven approach for parameter extraction. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3438–3444. IEEE, 2018.

[21] Anais Garrell, Luis Garza-Elizondo, Michael Villamizar, Fernando Herrero, and Alberto Sanfeliu. Aerial social force model: A new framework to accompany people using autonomous flying robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7011–7017. IEEE, 2017.

[22] Anais Garrell and Alberto Sanfeliu. Cooperative social robots to accompany groups of people. *The International Journal of Robotics Research*, 31(13):1675–1701, 2012.

[23] Atsushi Yamashita, Masaki Fukuchi, Jun Ota, Tamio Arai, and Hajime Asama. Motion planning for cooperative transportation of a large object by multiple mobile robots in a 3d environment. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 4, pages 3144–3151. IEEE, 2000.

[24] Erwin Prassler, Arno Ritter, Christoph Schaeffer, and Paolo Fiorini. A short history of cleaning robots. *Autonomous Robots*, 9(3):211–226, 2000.

[25] Frank Hegel, Claudia Muhl, Britta Wrede, Martina Hielscher-Fastabend, and Gerhard Sagerer. Understanding social robots. In *2009 Second International Conferences on Advances in Computer-Human Interactions*, pages 169–174. IEEE, 2009.

[26] Tony Belpaeme, James Kennedy, Aditi Ramachandran, Brian Scassellati, and Fumihide Tanaka. Social robots for education: A review. *Science robotics*, 3(21), 2018.

[27] Joost Broekens, Marcel Heerink, Henk Rosendal, et al. Assistive social robots in elderly care: a review. *Gerontechnology*, 8(2):94–103, 2009.

[28] Edward Twitchell Hall. *The hidden dimension*, volume 609. Garden City, NY: Doubleday, 1966.

[29] Ana del Valle Corrales Paredes and Miguel Ángel Salichs. *Sistema de navegación para robots sociales basado en señales*. PhD thesis, Universidad Carlos III de Madrid, 2012.

[30] Yoichi Morales, Takayuki Kanda, and Norihiro Hagita. Walking together: Side-by-side walking model for an interacting robot. *Journal of Human-Robot Interaction*, 3(2):50–73, 2014.

[31] Jorge Rios-Martinez, Anne Spalanzani, and Christian Laugier. From proxemics theory to socially-aware navigation: A survey. *International Journal of Social Robotics*, 7(2):137–153, 2015.

[32] Ely Repiso. Collaborative social robot navigation in accompanying and approaching tasks. 2020.

[33] Leila Takayama and Caroline Pantofaru. Influences on proxemic behaviors in human-robot interaction. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5495–5502. IEEE, 2009.

[34] Raja Chatila. Deliberation and reactivity in autonomous mobile robots. *Robotics and Autonomous Systems*, 16(2-4):197–211, 1995.

[35] Miguel Angel Salichs and Luis Moreno. Navigation of mobile robots: open questions. *Robotica*, 18(3):227–234, 2000.

[36] J Leonard. Mobile robot lacalization by tracking geometric beacons. *IEEE Trans. Robot. Autom.*, 7(3):89–97, 1991.

[37] Margrit Betke and Leonid Gurvits. Mobile robot localization using landmarks. *IEEE transactions on robotics and automation*, 13(2):251–263, 1997.

[38] Dieter Fox, Sebastian Thrun, Wolfram Burgard, and Frank Dellaert. Particle filters for mobile robot localization. In *Sequential Monte Carlo methods in practice*, pages 401–428. Springer, 2001.

[39] BK Patle, Anish Pandey, DRK Parhi, A Jagadeesh, et al. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606, 2019.

[40] Voemir Kunchev, Lakhmi Jain, Vladimir Ivancevic, and Anthony Finn. Path planning and obstacle avoidance for autonomous mobile robots: A review. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 537–544. Springer, 2006.

[41] Daniela Ridel, Eike Rehder, Martin Lauer, Christoph Stiller, and Denis Wolf. A literature review on the prediction of pedestrian behavior in urban scenarios. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3105–3112. IEEE, 2018.

[42] Purushothaman Raja and Sivagurunathan Pugazhenthi. Optimal path planning of mobile robots: A review. *International journal of physical sciences*, 7(9):1314–1320, 2012.

[43] Zahra Elmi and Mehmet Önder Efe. Multi-objective grasshopper optimization algorithm for robot path planning in static environments. In *2018 IEEE International Conference on Industrial Technology (ICIT)*, pages 244–249. IEEE, 2018.

[44] Marc Mehu and Klaus R Scherer. A psycho-ethological approach to social signal proces-

sing. *Cognitive processing*, 13(2):397–414, 2012.

[45] Kevin Toh. Social conventions: From language to law, 2010.

[46] Emrah Akin Sisbot, Luis F Marin-Urias, Xavier Broquere, Daniel Sidobre, and Rachid Alami. Synthesizing robot motions adapted to human presence. *International Journal of Social Robotics*, 2(3):329–343, 2010.

[47] Anais Garrell, Michael Villamizar, Francesc Moreno-Noguer, and Alberto Sanfeliu. Teaching robot's proactive behavior using human assistance. *International Journal of Social Robotics*, 9(2):231–249, 2017.

[48] Thibault Kruse, Patrizia Basili, Stefan Glasauer, and Alexandra Kirsch. Legible robot navigation in the proximity of moving humans. In *2012 IEEE workshop on advanced robotics and its social impacts (ARSO)*, pages 83–88. IEEE, 2012.

[49] Charles J Holahan. Environmental psychology. *Annual review of psychology*, 37(1):381–407, 1986.

[50] Silvia Casillas. Conducta espacial humana, 2009.

[51] Edward Twitchell Hall, Mildred Reed Hall, et al. *Understanding cultural differences*. Intercultural press, 1989.

[52] Carol Zinner Dolphin. Beyond hall: Variables in the use of personal space in intercultural transactions. *Howard Journal of Communications*, 1(1):23–38, 1988.

[53] John R Aiello, Gregory Nicosia, and Donna E Thompson. Physiological, social, and behavioral consequences of crowding on children and adolescents. *Child Development*, pages 195–202, 1979.

[54] National Academies of Sciences, Engineering, Medicine, et al. *How people learn II: Learners, contexts, and cultures*. National Academies Press, 2018.

[55] Eric Sundstrom and Irwin Altman. Interpersonal relationships and personal space: Research review and theoretical model. *Human Ecology*, 4(1):47–67, 1976.

[56] Mohammad Abu Yousuf, Yoshinori Kobayashi, Yoshinori Kuno, Akiko Yamazaki, and Keiichi Yamazaki. Development of a mobile museum guide robot that can configure spatial formation with visitors. In *International Conference on Intelligent Computing*, pages 423–432. Springer, 2012.

[57] Deneth Karunarathne, Yoichi Morales, Takayuki Kanda, and Hiroshi Ishiguro. Model of side-by-side walking without the robot knowing the goal. *International Journal of Social Robotics*, 10(4):401–420, 2018.

[58] Anaís Garrell, Carles Coll, René Alquézar, and Alberto Sanfeliu. Teaching a drone to accompany a person from demonstrations using non-linear asfm. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1985–1991. IEEE, 2019.

[59] Ely Repiso, Anaís Garrell, and Alberto Sanfeliu. People's adaptive side-by-side model

evolved to accompany groups of people by social robots. *IEEE Robotics and Automation Letters*, 5(2):2387–2394, 2020.

[60] Erwin Prassler, Dirk Bank, and Boris Kluge. Key technologies in robot assistants: Motion coordination between a human and a mobile robot. *Transactions on Control, Automation and Systems Engineering*, 4(1):56–61, 2002.

[61] Zhouxia Wang, Tianshui Chen, Jimmy Ren, Weihao Yu, Hui Cheng, and Liang Lin. Deep reasoning with knowledge graph for social relationship understanding. *arXiv preprint arXiv:1807.00504*, 2018.

[62] Qianru Sun, Bernt Schiele, and Mario Fritz. A domain based approach to social relation recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3481–3490, 2017.

[63] Junnan Li, Yongkang Wong, Qi Zhao, and Mohan S Kankanhalli. Visual social relationship recognition. *International Journal of Computer Vision*, 128(6):1750–1764, 2020.

[64] Zeynep Yucel, Francesco Zanlungo, Claudio Feliciani, Adrien Gregorj, and Takayuki Kanda. Estimating social relation from trajectories. *Collective Dynamics*, 5:222–229, 2020.

[65] Francesco Zanlungo, Zeynep Yücel, and Takayuki Kanda. Intrinsic group behaviour ii: On the dependence of triad spatial dynamics on social and personal features; and on the effect of social interaction on small group dynamics. *PloS one*, 14(12):e0225704, 2019.

[66] Alan P Fiske. The four elementary forms of sociality: framework for a unified theory of social relations. *Psychological review*, 99(4):689, 1992.

[67] Margaret S Clark and Judson Mills. Interpersonal attraction in exchange and communal relationships. *Journal of personality and social psychology*, 37(1):12, 1979.

[68] Edna B Foa and Uriel G Foa. Resource theory of social exchange. In *Handbook of social resource theory*, pages 15–32. Springer, 2012.

[69] Daphne Blunt Bugental. Acquisition of the algorithms of social life: a domain-based approach. *Psychological bulletin*, 126(2):187, 2000.

[70] Francesco Zanlungo, Dražen Brščić, and Takayuki Kanda. Spatial-size scaling of pedestrian groups under growing density conditions. *Physical Review E*, 91(6):062810, 2015.

[71] Charles Romesburg. *Cluster analysis for researchers*. Lulu. com, 2004.

[72] Eric Backer and Anil K Jain. A clustering performance measure based on fuzzy set decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1):66–75, 1981.

[73] Brian S Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. Cluster analysis 5th ed, 2011.

[74] Hana Řezanková and B Everitt. Cluster analysis and categorical data. *Statistika*, 89(3):216–232, 2009.

[75] Rui Xu and Don Wunsch. *Clustering*, volume 10. John Wiley & Sons, 2008.

[76] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[77] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[78] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017.

[79] Miguel A Carreira-Perpinán. A review of mean-shift algorithms for clustering. *arXiv preprint arXiv:1503.00687*, 2015.

[80] Konstantinos G Derpanis. Mean shift clustering. *Lecture Notes*, page 32, 2005.

[81] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[82] Ulrich Bodenhofer, Andreas Kothmeier, and Sepp Hochreiter. Apcluster: an r package for affinity propagation clustering. *Bioinformatics*, 27(17):2463–2464, 2011.

[83] Kaijun Wang, Junying Zhang, Dan Li, Xinna Zhang, and Tao Guo. Adaptive affinity propagation clustering. *arXiv preprint arXiv:0805.1096*, 2008.

[84] Zhaojuan Song. English speech recognition based on deep learning with multiple features. *Computing*, 102(3):663–682, 2020.

[85] Batuhan Balci, Dan Saadati, and Dan Shiferaw. Handwritten text recognition using deep learning. *CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, Course Project Report, Spring*, pages 752–759, 2017.

[86] Sheeba Joseph, R Sowmiya, Roshni Ann Thomas, and X Sofia. Face detection through neural network. In *Second International Conference on Current Trends In Engineering and Technology-ICCTET 2014*, pages 163–166. IEEE, 2014.

[87] Shaohua Wan, Lianyong Qi, Xiaolong Xu, Chao Tong, and Zonghua Gu. Deep learning models for real-time human activity recognition with smartphones. *Mobile Networks and Applications*, 25(2):743–755, 2020.

[88] Alexander Watson. Deep learning techniques for super-resolution in video games. *arXiv preprint arXiv:2012.09810*, 2020.

[89] MM Mehdy, PY Ng, EF Shair, NI Saleh, and Chandima Gomes. Artificial neural networks in image processing for early detection of breast cancer. *Computational and mathematical methods in medicine*, 2017, 2017.

[90] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[91] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–

444, 2015.

[92] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[93] Wikipedia. Aprendizaje profundo, 2021.

[94] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[95] Wenbin Yue, Zidong Wang, Hongwei Chen, Annette Payne, and Xiaohui Liu. Machine learning with applications in breast cancer diagnosis and prognosis. *Designs*, 2(2):13, 2018.

[96] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *Ieee access*, 6:35365–35381, 2018.

[97] Francisca Nonyelum Ogwueleka, Sanjay Misra, Ricardo Colomo-Palacios, and Luis Fernandez. Neural network and classification approach in identifying customer behavior in the banking sector: A case study of an international bank. *Human factors and ergonomics in manufacturing & service industries*, 25(1):28–42, 2015.

[98] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[99] Hussam Hatem Harz, Ahmed Osama Rafi, Musbah Osama Hijazi, and Samy S Abu-Naser. Artificial neural network for predicting diabetes using jnn. *International Journal of Academic Engineering Research (IJAER)*, 4(10), 2020.

[100] Tim P Vogels, Kanaka Rajan, and Larry F Abbott. Neural network dynamics. *Annu. Rev. Neurosci.*, 28:357–376, 2005.

[101] Jordi Torres. *DEEP LEARNING Introducción práctica con Keras*. Lulu.com, 2018.

[102] MY Rafiq, G Bugmann, and DJ Easterbrook. Neural network design for engineering applications. *Computers & Structures*, 79(17):1541–1552, 2001.

[103] Keun Young Lee, Namil Chung, and Suntae Hwang. Application of an artificial neural network (ann) model for predicting mosquito abundances in urban areas. *Ecological Informatics*, 36:172–180, 2016.

[104] Seyyed Reza Khaze, Mohammad Masdari, and Sohrab Hojjatkhah. Application of artificial neural networks in estimating participation in elections. *arXiv preprint arXiv:1309.2183*, 2013.

[105] Abhishek Pandey and Anu Mishra. Application of artificial neural networks in yield prediction of potato crop. *Russian Agricultural Sciences*, 43(3):266–272, 2017.

[106] Snehal S Dahikar and Sandeep V Rode. Agricultural crop yield prediction using artificial neural network approach. *International journal of innovative research in electrical, electronics, instrumentation and control engineering*, 2(1):683–686, 2014.

[107] Michal Turčaník. Packet filtering by artificial neural network. In *International Conference on Military Technologies (ICMT) 2015*, pages 1–4. IEEE, 2015.

[108] Kurt M Fanning and Kenneth O Cogger. Neural network detection of management fraud using published financial data. *Intelligent Systems in Accounting, Finance & Management*, 7(1):21–41, 1998.

[109] Mingyue Qiu, Yu Song, and Fumio Akagi. Application of artificial neural network for the prediction of stock market returns: The case of the japanese stock market. *Chaos, Solitons & Fractals*, 85:1–7, 2016.

[110] Aleksander Lotko, Paweł Albert Korneta, Małgorzata Anna Lotko, and Rafał Longwic. Using neural networks in modeling customer loyalty in passenger cars maintenance and repair services. *Applied Sciences*, 8(5):713, 2018.

[111] Hongjian Qi, Chen Chen, Haicang Zhang, John J Long, Wendy K Chung, Yongtao Guan, and Yufeng Shen. Mvp: predicting pathogenicity of missense variants by deep learning. *bioRxiv*, page 259390, 2018.

[112] Joseph Y Lo and CE Floyd. Application of artificial neural networks for diagnosis of breast cancer. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1755–1759. IEEE, 1999.

[113] Gustavo Santos-García and Emiliano Hernández Galilea. Using artificial neural networks to identify glaucoma stages. In *The Mystery of Glaucoma*, pages 331–352. InTech, 2011.

[114] Masaya Inoue, Sozo Inoue, and Takeshi Nishida. Deep recurrent neural network for mobile human activity recognition with high throughput. *Artificial Life and Robotics*, 23(2):173–185, 2018.

[115] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[116] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[117] Sanidhya Mangal, Rahul Modak, and Poorva Joshi. Lstm based music generation system. *arXiv preprint arXiv:1908.01080*, 2019.

[118] Abdullah Aziz Sharfuddin, Md Nafis Tihami, and Md Saiful Islam. A deep recurrent neural network with bilstm model for sentiment classification. In *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*, pages 1–4. IEEE, 2018.

[119] MK Vathsala and Ganga Holi. Rnn based machine translation and transliteration for twitter data. *International Journal of Speech Technology*, 23(3):499–504, 2020.

[120] Schalk Wilhelm Pienaar and Reza Malekian. Human activity recognition using lstm-rnn deep neural network architecture. In *2019 IEEE 2nd wireless africa conference (WAC)*, pages 1–5. IEEE, 2019.

[121] Sarfaraz Masood, Adhyan Srivastava, Harish Chandra Thuwal, and Musheer Ahmad. Real-time sign language gesture (word) recognition from video sequences using cnn and rnn. In *Intelligent Engineering Informatics*, pages 623–632. Springer, 2018.

[122] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5457–5466, 2018.

[123] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[124] Deepika Singh, Erinc Merdivan, Sten Hanke, Johannes Kropf, Matthieu Geist, and Andreas Holzinger. Convolutional and recurrent neural networks for activity recognition in smart environment. In *Towards integrative machine learning and knowledge extraction*, pages 194–205. Springer, 2017.

[125] Fangfang Yuan, Yanan Cao, Yanmin Shang, Yanbing Liu, Jianlong Tan, and Binxing Fang. Insider threat detection with deep neural network. In *International Conference on Computational Science*, pages 43–54. Springer, 2018.

[126] Sushanth Sreenivasa. Radial basis function (rbf) kernel: The go-to kernel, 2020.

Anexo A. Programa de extracción de datos básico

```
1
2  import pandas as pd
3  import numpy as np
4
5  import math
6  import glob
7  import csv
8  import os
9  import re
10
11 datos = np.array([])
12 resultados_var1 = []   # Lista donde guardar los parametros buscados
13 resultados_var1_file = []
14 acum_posx1 = 0
15 acum_posy1 = 0
16 acum_posz1 = 0
17 acum_posx2 = 0
18 acum_posy2 = 0
19 acum_posz2 = 0
20 acum_vel1 = 0
21 acum_velx1 = 0
22 acum_vely1 = 0
23 acum_vel2 = 0
24 acum_velx2 = 0
25 acum_vely2 = 0
26 acum_distancia = 0
27 acum_velocidad_relativa = 0
28 acum_velocidad_grupo = 0
29 elem = 0
30 dic1 = ["Interacting", "Not_interacting"]                    # Lista de las
       categorias principales
31 dic2 = ["Colleagues", "Couples", "Families", "Friends"]     # Lista de las
       categorias secundarias
32
33 for x in dic1:                      # Recorre la lista de categorias principales
34     for y in dic2:                  # Recorre la lista de categorias secundarias
35         list_of_files = glob.glob('./Datos/UPC/%s/%s/*.dat' % (x,y))  # Lista de
        los archivos .dat de la carpeta
36         list_of_files = sorted(list_of_files)     # Ordena los archivos segun el
        orden String
37         orden = 0                        # Variable para almacenar la posicion
       del archivo en la lista
38         cent = '100'
39         list_cent = []
40         day_new = ''      # Variable para almacenar los nuevos dias (primer
       numero de los archivos)
41         num_new = ''      # Variable para almacenar los nuevos numeros (segundo
       numero de los archivos)
42
43         # For para ordenar num ricamente los archivos de la lista:
44         for k in range(len(list_of_files)):     # Recorre los archivos de la
       lista
45             latest_file = list_of_files[k]      # Selecciona el archivo actual
46             barra = 0                           # Resetea la variable barra que
       cuenta las "_" del nombre de los archivos
47             day_old = day_new                   # Actualiza day_old
48             num_old = num_new                   # Actualiza num_old
```

```
            day_new = ''                        # Resetea day_new
            num_new = ''                        # Resetea num_new
            for w in range(len(latest_file)):   # Recorre los strings del nombre del archivo actual
                if latest_file[w] == '_':       # Si el string es una "_":
                    barra += 1                  # Actualiza la variable barra
                    if (x == dic1[0] and barra == 1) or (x == dic1[1] and barra==2):  # Si ya ha contado 1 barra:
                        for m in range(len(latest_file)):
                            if latest_file[w+m+1] != '_':        # Avanza hasta encontrar otra "_"
                                day_new += latest_file[w+m+1]    # Guarda el dia del archivo
                            else:
                                break
                    if (x == dic1[0] and barra == 2) or (x == dic1[1] and barra==3):  # Si ya ha contado 2 barras:
                        for m in range(len(latest_file)):
                            if latest_file[w+m+1] != '.':         # Avanza hasta encontrar un "."
                                num_new += latest_file[w+m+1]     # Guarda el numero del archivo
                            else:
                                break
            if num_new == '1':                          # Si el numero del archivo es "1":
                orden = k                               # Guarda la posicion del archivo en la lista
                cent = '100'
                list_cent = []
            if day_old == day_new and len(num_new) == 1:        # Si el dia del archivo anterior es igual al actual y el d a actual es de un d gito:
                list_of_files.remove(latest_file)               # Elimina el archivo actual
                list_of_files.insert(orden + 1, latest_file)    # Inserta el archivo actual en el sitio correspondiente
                orden += 1                          # Actualiza la variable de posicion

            # Si los numeros de los archivos son > 100 hacen unos pasos extras:
            if num_new == cent:
                list_cent.append(latest_file)
                cent = str((int(cent)+1))
            if num_new == '99' and  len(list_cent) != 0:
                for m in range(len(list_cent)):
                    list_of_files.remove(list_cent[m])
                    list_of_files.insert(k, list_cent[m])

        days = []       # Lista para guardar los dias de los archivos
        nums = []       # Lista para guardar los numeros de los archivos
        day_new = ''    # Resetea la variable day_new para actualizar day_old en el siguiente For
        num_new = 0     # Inicia la variable num_new para actualizar num_old en el siguiente For

        # For para completar las listas days y nums:
        for k in range(len(list_of_files)):     # Recorre los archivos de la lista
            latest_file = list_of_files[k]      # Selecciona el archivo actual
            barra = 0   # Resetea la variable barra que cuenta las "_" del
```

```
nombre de los archivos
93          day_old = day_new         # Actualiza day_old
94          num_old = num_new         # Actualiza num_old
95          day_new = ''              # Resetea day_new
96          num_new = ''              # Resetea num_new
97          for w in range(len(latest_file)):    # Recorre los strings del
nombre del archivo actual
98              if latest_file[w] == '_':        # Si el string es una "_":
99                  barra += 1                   # Actualiza la variable barra
100                 if (x == dic1[0] and barra == 1) or (x == dic1[1] and barra
==2):   # Si ya ha contado 1 barra:
101                     for m in range(len(latest_file)):
102                         if latest_file[w+m+1] != '_':        # Avanza hasta
encontrar otra "_"
103                             day_new += latest_file[w+m+1]    # Guarda el dia
 del archivo
104                         else:
105                             break
106                 if (x == dic1[0] and barra == 2) or (x == dic1[1] and barra
==3):   # Si ya ha contado 2 barras:
107                     for m in range(len(latest_file)):
108                         if latest_file[w+m+1] != '.':        # Avanza hasta
encontrar un "."
109                             num_new += latest_file[w+m+1]    # Guarda el
numero del archivo
110                         else:
111                             break
112         num_new = int(num_new)             # Convierte num_new en un entero
113         if day_old != day_new:             # Si el dia actual es diferente
del anterior:
114             days.append(day_new)           # Anade el dia actual a la lista
days
115         if num_old >= num_new:             # Si el n mero anterior es mayor
 al actual:
116             nums.append(num_old)           # Anade el n mero anterior a la
lista nums
117         if (k+1) == len(list_of_files):    # Si la siguiente posicion es la
ultima de la lista:
118             nums.append(num_new)           # Anade el numero actual  a la
lista nums
119     print(days)
120     print(nums)
121
122     # For para recorrer todos los archivos de las carpetas:
123     for i in days:                         # Recorre los elementos de la
lista days
124         ind = days.index(i)
125         for j in range(nums[ind]):         # Recorre los elementos de la
lista nums
126             with open("Datos/UPC/%s/%s/day_%s_%d.dat" % (x, y, i, j+1), "r")
 as f:   # Abre el archivo
127                 # with open("Datos/UPC/Interacting/Colleagues/day_0109_1.dat
", "r") as f:
128                 # print("Datos/UPC/%s/%s/day_%s_%d.dat" % (x, y, i, j+1))
129                 count = 0     # Variable para contar cuantas l neas del
archivo ha recorrido
130                 for line in f:    # Recorre cada l nea del archivo
131                     tempo = 0     # Variable para contar los espacios en
blanco " " de la l nea
132                     count += 1    # Actualiza la variable count
```

```
                        if count <= 6:      # Si est s en alguna de las 6 primeras l neas:
                            pass            # Pasa
                        else:               # Si has pasado las 6 primeras lineas:
                            for k in range(len(line)):   # Recorre los strings de la linea actual:
                                if line[k].isspace():    # Si es un " ":
                                    if tempo == 0:
                                        time = ''
                                        for m in range(k+1):
                                            if line[m] != ' ':    # Avanza hasta encontrar otro " ":
                                                time += line[m]   # Guarda la variable time
                                            else:
                                                break
                                    tempo += 1              # Actualiza la variable tempo
                                    if tempo == 2:          # Si has contado dos espacios en la linea actual:
                                        posx1 = ''          # Inicializa la variable posx1
                                        for m in range(len(line)):
                                            if line[k+m+1] != ' ':   # Avanza hasta encontrar otro " ":
                                                posx1 += line[k+m+1]   # Guarda la variable posx1
                                            else:
                                                break
                                    elif tempo == 3:         # Si has contado tres espacios en la linea actual:
                                        posy1 = ''           # Inicializa la variable posy1
                                        for m in range(len(line)):
                                            if line[k+m+1] != ' ':   # Avanza hasta encontrar otro " ":
                                                posy1 += line[k+m+1]  # Guarda la variable posy1
                                            else:
                                                break
                                    elif tempo == 4:   # Si has contado cuatro espacios en la linea actual:
                                        posz1 = ''    # Inicializa la variable posz1
                                        for m in range(len(line)):
                                            if line[k+m+1] != ' ':    # Avanza hasta encontrar otro " ":
                                                posz1 += line[k+m+1]  # Guarda la variable posz1
                                            else:
                                                break
                                    elif tempo == 5:   # Si has contado cinco espacios en la linea actual:
                                        velx1 = ''     # Inicializa la variable velx1
                                        for m in range(len(line)):
                                            if line[k+m+1] != ' ':    # Avanza hasta encontrar otro " ":
                                                velx1 += line[k+m+1]  # Guarda
```

```
la variable velx1
                                        else:
                                            break
                                elif tempo == 6:    # Si has contado seis
espacios en la linea actual:
                                    vely1 = ''      # Inicializa la variable
 posy1
                                    for m in range(len(line)):
                                        if line[k+m+1] != ' ':    # Avanza
hasta encontrar otro " ":
                                            vely1 += line[k+m+1]  # Guarda
la variable vely1
                                        else:
                                            break
                                elif tempo == 7:   # Si has contado siete
espacios en la linea actual:
                                    vel1 = ''      # Inicializa la variable
vel1
                                    for m in range(len(line)):
                                        if line[k+m+1] != ' ':   # Avanza
hasta encontrar otro " ":
                                            vel1 += line[k+m+1]  # Guarda la
 variable vel1
                                        else:
                                            break
                                elif tempo == 9:     # Si has contado nueve
espacios en la linea actual:
                                    posx2 = ''       # Inicializa la
variable posx2
                                    for m in range(len(line)):
                                        if line[k+m+1] != ' ':    # Avanza
hasta encontrar otro " ":
                                            posx2 += line[k+m+1]  # Guarda
la variable posx2
                                        else:
                                            break
                                elif tempo == 10:     # Si has contado diez
espacios en la linea actual:
                                    posy2 = ''        # Inicializa la
variable posy2
                                    for m in range(len(line)):
                                        if line[k+m+1] != ' ':    # Avanza
hasta encontrar otro " ":
                                            posy2 += line[k+m+1]  # Guarda
la variable posy2
                                        else:
                                            break
                                elif tempo == 11:      # Si has contado once
 espacios en la linea actual:
                                    posz2 = ''         # Inicializa la
variable posz2
                                    for m in range(len(line)):
                                        if line[k+m+1] != ' ':    # Avanza
hasta encontrar otro " ":
                                            posz2 += line[k+m+1]  # Guarda
la variable posz2
                                        else:
                                            break
                                elif tempo == 12:       # Si has contado
doce espacios en la linea actual:
```

```
                                        velx2 = ''          # Inicializa la variable velx2
                                        for m in range(len(line)):
                                            if line[k+m+1] != ' ':   # Avanza hasta encontrar otro " ":
                                                velx2 += line[k+m+1] # Guarda la variable velx2
                                            else:
                                                break
                                    elif tempo == 13:       # Si has contado trece espacios en la linea actual:
                                        vely2 = ''          # Inicializa la variable vely2
                                        for m in range(len(line)):
                                            if line[k+m+1] != ' ':   # Avanza hasta encontrar otro " ":
                                                vely2 += line[k+m+1] # Guarda la variable vely2
                                            else:
                                                break
                                    elif tempo == 14:       # Si has contado catorce espacios en la linea actual:
                                        vel2 = ''           # Inicializa la variable vel2
                                        for m in range(len(line)):
                                            if line[k+m+1] != ' ':   # Avanza hasta encontrar otro " ":
                                                vel2 += line[k+m+1]  # Guarda la variable vel2
                                            else:
                                                break
                                        if count == 7:
                                            time_start = float(time)  # Convierte las variables string en float
                                        elif count > 7:
                                            time_end = float(time)
                                        posx1 = float(posx1)
                                        posy1 = float(posy1)
                                        posz1 = float(posz1)
                                        posx2 = float(posx2)
                                        posy2 = float(posy2)
                                        posz2 = float(posz2)
                                        vel1 = float(vel1)
                                        velx1 = float(velx1)
                                        vely1 = float(vely1)
                                        vel2 = float(vel2)
                                        velx2 = float(velx2)
                                        vely2 = float(vely2)
                                        distancia = math.sqrt(pow(posx2 - posx1, 2) + pow(posy2 - posy1, 2))   # Calcula la distancia euclidiana
                                        velocidad_relativa = abs(vel1 - vel2) # Calcula la velocidad relativa
                                        velocidad_grupo = (vel1 + vel2) / 2 # Calcula la velocidad de grupo

                                        acum_posx1 += posx1     # Acumulador de las variables
                                        acum_posy1 += posy1
                                        acum_posz1 += posz1
                                        acum_posx2 += posx2
```

```
                                        acum_posy2 += posy2
                                        acum_posz2 += posz2
                                        acum_vel1 += vel1
                                        acum_velx1 += velx1
                                        acum_vely1 += vely1
                                        acum_vel2 += vel2
                                        acum_velx2 += velx2
                                        acum_vely2 += vely2
                                        acum_distancia += distancia
                                        acum_velocidad_relativa +=
velocidad_relativa
                                        acum_velocidad_grupo += velocidad_grupo

                    if count > 6:
                        count = count - 6
                        total_time = time_end - time_start
                        total_posx1 = acum_posx1 / count        # Media de las
variables
                        total_posy1 = acum_posy1 / count
                        total_posz1 = acum_posz1 / count
                        total_posx2 = acum_posx2 / count
                        total_posy2 = acum_posy2 / count
                        total_posz2 = acum_posz2 / count
                        total_vel1 = acum_vel1 / count
                        total_velx1 = acum_velx1 / count
                        total_vely1 = acum_vely1 / count
                        total_vel2 = acum_vel2 / count
                        total_velx2 = acum_velx2 / count
                        total_vely2 = acum_vely2 / count
                        total_distancia = acum_distancia / count    # Media de
la distancia euclidiana
                        total_velocidad_relativa = acum_velocidad_relativa /
count    # Media de la velocidad relativa
                        total_velocidad_grupo = acum_velocidad_grupo / count
 # Media de la velocidad de grupo

                        if y == "Colleagues":    # Si pertenece a la categoria
Colegas
                            label = 0            # Clasifica en la categoria 0
                        elif y == "Couples":     # Si pertenece a la categoria
Pareja
                            label = 1            # Clasifica en la categoria 1
                        elif y == "Families":    # Si pertenece a la categoria
Familia
                            label = 2            # Clasifica en la categoria 2
                        elif y == "Friends":     # Si pertenece a la categoria
Amistad
                            label = 3            # Clasifica en la categoria 3


                    #####

                        resultados_var1.append([label, total_time, total_posx1,
total_posy1, total_posz1, total_velx1, total_vely1, total_vel1, total_posx2,
 total_posy2, total_posz2, total_velx2, total_vely2, total_vel2,
total_distancia, total_velocidad_relativa, total_velocidad_grupo])   # Anade
 a la lista resultados
                        resultados_var1_file.append([label, total_time,
total_posx1, total_posy1, total_posz1, total_velx1, total_vely1, total_vel1,
 total_posx2, total_posy2, total_posz2, total_velx2, total_vely2, total_vel2
```

```
    , total_distancia, total_velocidad_relativa, total_velocidad_grupo])

                    #####

                    acum_posx1 = 0
                    acum_posy1 = 0
                    acum_posz1 = 0
                    acum_posx2 = 0
                    acum_posy2 = 0
                    acum_posz2 = 0
                    acum_vel1 = 0
                    acum_velx1 = 0
                    acum_vely1 = 0
                    acum_vel2 = 0
                    acum_velx2 = 0
                    acum_vely2 = 0
                    acum_distancia = 0
                    acum_velocidad_relativa = 0
                    acum_velocidad_grupo = 0
                    elem += 1

        #####

        # Crea un csv con los resultados dentro las mismas carpetas de los archivos .dat
        with open("Datos/UPC/%s/%s/%s_%s_data_Clustering_Temporal_1.csv" % (x, y, x, y), "w", newline = "") as file:  # Crea un nuevo archivo csv
            writer = csv.writer(file)
            writer.writerows(resultados_var1)    # Inserta los elementos de la lista resultados en cada fila del nuevo archivo

        # Crea un csv con los resultados en carpetas diferentes
        with open("Resultados/%s/%s/%s_%s_data_Clustering_Temporal_1.csv" % (x, y, x, y), "w", newline = "") as file:  # Crea un nuevo archivo csv
            writer = csv.writer(file)
            writer.writerows(resultados_var1)    # Inserta los elementos de la lista resultados en cada fila del nuevo archivo
        resultados_var1 = []                     # Resetea la lista resultados


        if (x == "Interacting" and y == "Friends") or (x == "Not_interacting" and y == "Friends"):

            # Crea un csv con los resultados en carpetas diferentes
            with open("Resultados/%s/%s_data_Clustering_Temporal_1.csv" % (x, x), "w", newline = "") as file:  # Crea un nuevo archivo csv
                writer = csv.writer(file)
                writer.writerows(resultados_var1_file)     # Inserta los elementos de la lista resultados en cada fila del nuevo archivo
            resultados_var1_file = []                      # Resetea la lista resultados

        # Pasa a la siguiente carpeta secundaria
    # Pasa a la siguiente carpeta principal

print("Datos obtenidos de %d elementos" % elem)
```

Anexo B. Programa de clasificación basado en *clustering*

```
1
2  from sklearn.cluster import KMeans
3  from sklearn.cluster import SpectralClustering
4  from sklearn.cluster import AgglomerativeClustering
5  from sklearn.cluster import MeanShift, estimate_bandwidth
6  from sklearn.cluster import AffinityPropagation
7
8  from sklearn import metrics
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 import csv
13 import sys
14
15 np.set_printoptions(threshold=sys.maxsize)
16
17 from itertools import cycle
18 from mpl_toolkits.mplot3d import Axes3D
19
20 dic1 = ["Interacting", "Not_interacting"]                    # Lista de las
       categorias principales
21 dic2 = ["Colleagues", "Couples", "Families", "Friends"]     # Lista de las
       categorias secundarias
22
23 count = 0
24
25 # Recorre los archivos creados por el programa de extraccion de datos
26
27 for x in dic1:
28     for y in dic2:
29         with open("Resultados/%s/%s/%s_%s_data_Clustering_Temporal_3.csv" % (x,
       y, x, y), "r") as f:   # Abre el archivo
30         # with open("Resultados/Interacting/Colleagues/
       Interacting_Colleagues_data_Clustering_Temporal_2.csv", "r") as f:
31             reader = csv.reader(f, delimiter = ',')
32             globals()['datos_%s_%s' % (x, y)]= np.array(list(reader)).astype(
       float)
33             if count == 0:
34                 datos_todos = globals()['datos_%s_%s' % (x, y)]
35                 count += 1
36             else:
37                 datos_todos = np.append(datos_todos, globals()['datos_%s_%s' % (
       x, y)], axis = 0)
38                 count += 1
39             if count == 4:
40                 datos_Interacting = datos_todos
41             elif count == 5:
42                 datos_Not_interacting = globals()['datos_%s_%s' % (x, y)]
43             elif count > 5:
44                 datos_Not_interacting = np.append(datos_Not_interacting, globals
       ()['datos_%s_%s' % (x, y)], axis = 0)
45
46
47 ################################################################
48
49 ##### Metodo K-Means
50
```

```
kmeans = KMeans(n_clusters=4, random_state=0).fit(datos_Interacting)
labels_km = kmeans.labels_

##### Calculo del error

#print('Labels True')
#print(datos_Interacting[:, 0])
#print('Labels Predicted')
#print(labels_km)

print("##################################################")

print("Accuracy K-Means")

accuracy_km_ARI = metrics.adjusted_rand_score(datos_Interacting[:, 0], labels_km)
print("Accuracy K-Means - Adjusted Rand Index: %d" % accuracy_km_ARI)

accuracy_km_AMI = metrics.adjusted_mutual_info_score(datos_Interacting[:, 0], labels_km)
print("Accuracy K-Means - Mutual Information based scores: %d" % accuracy_km_AMI)

accuracy_km_VM = metrics.v_measure_score(datos_Interacting[:, 0], labels_km)
print("Accuracy K-Means - V-measure: %d" % accuracy_km_VM)

accuracy_km_FMI = metrics.fowlkes_mallows_score(datos_Interacting[:, 0], labels_km)
print("Accuracy K-Means - Fowlkes Mallows scores: %d" % accuracy_km_FMI)

km_CM = metrics.confusion_matrix(datos_Interacting[:, 0], labels_km)
print("K-Means - Confusion Matrix:")
print(km_CM)
accuracy_km_CM = ((km_CM[0][0]+km_CM[1][1]+km_CM[2][2]+km_CM[3][3])/(km_CM[0][0]+km_CM[0][1]+km_CM[0][2]+km_CM[0][3]+km_CM[1][0]+km_CM[1][1]+km_CM[1][2]+km_CM[1][3]+km_CM[2][0]+km_CM[2][1]+km_CM[2][2]+km_CM[2][3]+km_CM[3][0]+km_CM[3][1]+km_CM[3][2]+km_CM[3][3]))*100
print("Accuracy K-Means - Confusion Matrix: %d %%" % accuracy_km_CM)


##### K-Means 3D plot

fig = plt.figure(1, figsize=(4,3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(datos_Interacting[:, 2], datos_Interacting[:, 3], datos_Interacting[:, 4], c=labels_km.astype(float), edgecolor='k')
ax.set_xlabel('Distancia', labelpad=20)
ax.set_ylabel('Vel. Relativa', labelpad=20)
ax.set_zlabel('Vel. Grupo', labelpad=20)
ax.set_title('K-Means 3D')
ax.dist = 12


##################################################################

##### Metodo Spectral Clustering

clustering_sp = SpectralClustering(n_clusters=4, random_state=0).fit(datos_Interacting)
labels_sp = clustering_sp.labels_
```

```
102
103 ##### Calculo del error
104
105 print("##############################################")
106
107 print("Accuracy Spectral Clustering")
108
109 accuracy_sp_ARI = metrics.adjusted_rand_score(datos_Interacting[:, 0], labels_sp
    )
110 print("Accuracy Spectral Clustering - Adjusted Rand Index: %d" % accuracy_sp_ARI
    )
111
112 accuracy_sp_AMI = metrics.adjusted_mutual_info_score(datos_Interacting[:, 0],
    labels_sp)
113 print("Accuracy Spectral Clustering - Mutual Information based scores: %d" %
    accuracy_sp_AMI)
114
115 accuracy_sp_VM = metrics.v_measure_score(datos_Interacting[:, 0], labels_sp)
116 print("Accuracy Spectral Clustering - V-measure: %d" % accuracy_sp_VM)
117
118 accuracy_sp_FMI = metrics.fowlkes_mallows_score(datos_Interacting[:, 0],
    labels_sp)
119 print("Accuracy Spectral Clustering - Fowlkes Mallows scores: %d" %
    accuracy_sp_FMI)
120
121 sp_CM = metrics.confusion_matrix(datos_Interacting[:, 0], labels_sp)
122 print("Spectral Clustering - Confusion Matrix:")
123 print(sp_CM)
124 accuracy_sp_CM = ((sp_CM[0][0]+sp_CM[1][1]+sp_CM[2][2]+sp_CM[3][3])/(sp_CM
    [0][0]+sp_CM[0][1]+sp_CM[0][2]+sp_CM[0][3]+sp_CM[1][0]+sp_CM[1][1]+sp_CM
    [1][2]+sp_CM[1][3]+sp_CM[2][0]+sp_CM[2][1]+sp_CM[2][2]+sp_CM[2][3]+sp_CM
    [3][0]+sp_CM[3][1]+sp_CM[3][2]+sp_CM[3][3]))*100
125 print("Accuracy Spectral Clustering - Confusion Matrix: %d %%" % accuracy_sp_CM)
126
127
128 ##### Spectral Clustering 3D plot
129
130 fig = plt.figure(2, figsize=(4,3))
131 ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
132 ax.scatter(datos_Interacting[:, 2], datos_Interacting[:, 3], datos_Interacting
    [:, 4], c=labels_sp.astype(float), edgecolor='k')
133 ax.set_xlabel('Distancia', labelpad=20)
134 ax.set_ylabel('Vel. Relativa', labelpad=20)
135 ax.set_zlabel('Vel. Grupo', labelpad=20)
136 ax.set_title('Spectral Clustering 3D')
137 ax.dist = 12
138
139
140 ##############################################################
141
142 ##### Metodo Hierarchical clustering (Agglomerative Clustering)
143
144 #clustering_hc = AgglomerativeClustering().fit(datos_Interacting)
145 clustering_hc = AgglomerativeClustering(n_clusters=4).fit(datos_Interacting)
146 labels_hc = clustering_hc.labels_
147
148 ##### Calculo del error
149
150 print("##############################################")
151
```

```
print("Accuracy Hierarchical clustering")

accuracy_hc_ARI = metrics.adjusted_rand_score(datos_Interacting[:, 0], labels_hc
    )
print("Accuracy Hierarchical clustering - Adjusted Rand Index: %d" %
    accuracy_hc_ARI)

accuracy_hc_AMI = metrics.adjusted_mutual_info_score(datos_Interacting[:, 0],
    labels_hc)
print("Accuracy Hierarchical clustering - Mutual Information based scores: %d" %
     accuracy_hc_AMI)

accuracy_hc_VM = metrics.v_measure_score(datos_Interacting[:, 0], labels_hc)
print("Accuracy Hierarchical clustering - V-measure: %d" % accuracy_hc_VM)

accuracy_hc_FMI = metrics.fowlkes_mallows_score(datos_Interacting[:, 0],
    labels_hc)
print("Accuracy Hierarchical clustering - Fowlkes Mallows scores: %d" %
    accuracy_hc_FMI)

hc_CM = metrics.confusion_matrix(datos_Interacting[:, 0], labels_hc)
print("Hierarchical clustering - Confusion Matrix:")
print(hc_CM)
accuracy_hc_CM = ((hc_CM[0][0]+hc_CM[1][1]+hc_CM[2][2]+hc_CM[3][3])/(hc_CM
    [0][0]+hc_CM[0][1]+hc_CM[0][2]+hc_CM[0][3]+hc_CM[1][0]+hc_CM[1][1]+hc_CM
    [1][2]+hc_CM[1][3]+hc_CM[2][0]+hc_CM[2][1]+hc_CM[2][2]+hc_CM[2][3]+hc_CM
    [3][0]+hc_CM[3][1]+hc_CM[3][2]+hc_CM[3][3]))*100
print("Accuracy Hierarchical clustering - Confusion Matrix: %d %%" %
    accuracy_hc_CM)

##### Hierarchical clustering (Agglomerative Clustering) 3D plot

fig = plt.figure(3, figsize=(4,3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(datos_Interacting[:, 2], datos_Interacting[:, 3], datos_Interacting
    [:, 4], c=labels_hc.astype(float), edgecolor='k')
ax.set_xlabel('Distancia', labelpad=20)
ax.set_ylabel('Vel. Relativa', labelpad=20)
ax.set_zlabel('Vel. Grupo', labelpad=20)
ax.set_title('Hierarchical clustering 3D')
ax.dist = 12


#######################################################################

##### Metodo Mean-shift

bandwidth = estimate_bandwidth(datos_Interacting, quantile=0.3)
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
#ms = MeanShift(bandwidth=bandwidth, bin_seeding=True, max_iter=100)
ms_fit = ms.fit(datos_Interacting)
labels_ms = ms.labels_
cluster_centers = ms.cluster_centers_

labels_unique = np.unique(labels_ms)
n_clusters_ms = len(labels_unique)

print("##################################################")

print("Number of estimated Mean-shift clusters: %d" % n_clusters_ms)
```

```
201
202 ##### Calculo del error
203
204 print("Accuracy Mean-shift Clustering")
205
206 accuracy_ms_ARI = metrics.adjusted_rand_score(datos_Interacting[:, 0], labels_ms
    )
207 print("Accuracy Mean-shift - Adjusted Rand Index: %d" % accuracy_ms_ARI)
208
209 accuracy_ms_AMI = metrics.adjusted_mutual_info_score(datos_Interacting[:, 0],
    labels_ms)
210 print("Accuracy Mean-shift - Mutual Information based scores: %d" %
    accuracy_ms_AMI)
211
212 accuracy_ms_VM = metrics.v_measure_score(datos_Interacting[:, 0], labels_ms)
213 print("Accuracy Mean-shift - V-measure: %d" % accuracy_ms_VM)
214
215 accuracy_ms_FMI = metrics.fowlkes_mallows_score(datos_Interacting[:, 0],
    labels_ms)
216 print("Accuracy Mean-shift - Fowlkes Mallows scores: %d" % accuracy_ms_FMI)
217
218 #ms_CM = metrics.confusion_matrix(datos_Interacting[:, 0], labels_ms)
219 #print("Mean-shift - Confusion Matrix:")
220 #print(ms_CM)
221 #accuracy_ms_CM = ((ms_CM[0][0]+ms_CM[1][1]+ms_CM[2][2]+ms_CM[3][3])/(ms_CM
    [0][0]+ms_CM[0][1]+ms_CM[0][2]+ms_CM[0][3]+ms_CM[1][0]+ms_CM[1][1]+ms_CM
    [1][2]+ms_CM[1][3]+ms_CM[2][0]+ms_CM[2][1]+ms_CM[2][2]+ms_CM[2][3]+ms_CM
    [3][0]+ms_CM[3][1]+ms_CM[3][2]+ms_CM[3][3]))*100
222 #print("Accuracy Mean-shift - Confusion Matrix: %d %%" % accuracy_ms_CM)
223
224 ##### Mean-shift 3D plot
225
226 fig = plt.figure(4, figsize=(4,3))
227 ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
228 ax.scatter(datos_Interacting[:, 2], datos_Interacting[:, 3], datos_Interacting
    [:, 4], c=labels_ms.astype(float), edgecolor='k')
229 ax.set_xlabel('Distancia', labelpad=20)
230 ax.set_ylabel('Vel. Relativa', labelpad=20)
231 ax.set_zlabel('Vel. Grupo', labelpad=20)
232 ax.set_title('Mean-shift 3D')
233 ax.dist = 12
234
235 ######################################################################
236
237 ##### Metodo Affinity Propagation clustering algorithm
238
239 af = AffinityPropagation().fit(datos_Interacting)
240 #af = AffinityPropagation(preference=-50).fit(datos_Interacting)
241 cluster_centers_indices = af.cluster_centers_indices_
242 labels_af = af.labels_
243
244 n_clusters_af = len(cluster_centers_indices)
245
246 print("##################################################")
247
248 print('Estimated number of Affinity clusters: %d' % n_clusters_af)
249
250 ##### Calculo del error
251
252 print("Accuracy Affinity Propagation Clustering")
```

```python

accuracy_af_ARI = metrics.adjusted_rand_score(datos_Interacting[:, 0], labels_af)
print("Accuracy Affinity Propagation - Adjusted Rand Index: %d" % accuracy_af_ARI)

accuracy_af_AMI = metrics.adjusted_mutual_info_score(datos_Interacting[:, 0], labels_af)
print("Accuracy Affinity Propagation - Mutual Information based scores: %d" % accuracy_af_AMI)

accuracy_af_VM = metrics.v_measure_score(datos_Interacting[:, 0], labels_af)
print("Accuracy Affinity Propagation - V-measure: %d" % accuracy_af_VM)

accuracy_af_FMI = metrics.fowlkes_mallows_score(datos_Interacting[:, 0], labels_af)
print("Accuracy Affinity Propagation - Fowlkes Mallows scores: %d" % accuracy_af_FMI)

#af_CM = metrics.confusion_matrix(datos_Interacting[:, 0], labels_af)
#print("Affinity Propagation - Confusion Matrix:")
#print(af_CM)
#accuracy_af_CM = ((af_CM[0][0]+af_CM[1][1]+af_CM[2][2]+af_CM[3][3])/(af_CM[0][0]+af_CM[0][1]+af_CM[0][2]+af_CM[0][3]+af_CM[1][0]+af_CM[1][1]+af_CM[1][2]+af_CM[1][3]+af_CM[2][0]+af_CM[2][1]+af_CM[2][2]+af_CM[2][3]+af_CM[3][0]+af_CM[3][1]+af_CM[3][2]+af_CM[3][3]))*100
#print("Accuracy Affinity Propagation - Confusion Matrix: %d %%" % accuracy_af_CM)

##### Affinity Propagation clustering algorithm 3D plot

fig = plt.figure(5, figsize=(4,3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(datos_Interacting[:, 2], datos_Interacting[:, 3], datos_Interacting[:, 4], c=labels_af.astype(float), edgecolor='k')
ax.set_xlabel('Distancia', labelpad=20)
ax.set_ylabel('Vel. Relativa', labelpad=20)
ax.set_zlabel('Vel. Grupo', labelpad=20)
ax.set_title('Affinity Propagation 3D')
ax.dist = 12
plt.show()

#####################################################################
```

Anexo C. Programa de clasificación utilizando una red neuronal estándar

```
1
2  import math
3  import numpy as np
4  import h5py
5  import matplotlib.pyplot as plt
6  from sklearn.model_selection import train_test_split
7  import csv
8  import sys
9
10 import tensorflow as tf
11 from tensorflow import keras
12 from tensorflow.python.framework import ops
13
14 ############### Preparaci n de los datos
       ##########################################################
15
16 np.set_printoptions(threshold=sys.maxsize)
17
18 dic1 = ["Interacting", "Not_interacting"]                    # Lista de las
       categorias principales
19 dic2 = ["Colleagues", "Couples", "Families", "Friends"]    # Lista de las
       categorias secundarias
20
21 count = 0
22
23 # Recorre los archivos creados por el programa de extraccion de datos para
       guardar las variables
24
25 for x in dic1:
26     for y in dic2:
27         with open("Resultados/%s/%s/%s_%s_data_NN_Temporal_3.csv" % (x, y, x, y)
       , "r") as f: # Abre el archivo
28         # with open("Resultados/Interacting/Colleagues/
       Interacting_Colleagues_data_NN_Temporal_3.csv", "r") as f:
29             reader = csv.reader(f, delimiter = ',')
30             globals()['datos_%s_%s' % (x, y)]= np.array(list(reader)).astype(
       float)
31             if count == 0:
32                 datos_todos = globals()['datos_%s_%s' % (x, y)]
33                 count += 1
34             else:
35                 datos_todos = np.append(datos_todos, globals()['datos_%s_%s' % (
       x, y)], axis = 0)
36                 count += 1
37             if count == 4:
38                 datos_Interacting = datos_todos
39             elif count == 5:
40                 datos_Not_interacting = globals()['datos_%s_%s' % (x, y)]
41             elif count > 5:
42                 datos_Not_interacting = np.append(datos_Not_interacting, globals
       ()['datos_%s_%s' % (x, y)], axis = 0)
43
44 count = 0
45
46 # Recorre los archivos creados por el programa de extraccion de datos para
```

```
    guardar las categorias

for x in dic1:
    for y in dic2:
        with open("Resultados/%s/%s/%s_%s_data_NN_Temporal_Labels.csv" % (x, y, x, y), "r") as f:   # Abre el archivo
        # with open("Resultados/Interacting/Colleagues/Interacting_Colleagues_data_NN_Temporal_Labels.csv", "r") as f:
            reader = csv.reader(f, delimiter = ',')
            globals()['labels_%s_%s' % (x, y)]= np.array(list(reader)).astype(int)
            if count == 0:
                labels_todos = globals()['labels_%s_%s' % (x, y)]
                count += 1
            else:
                labels_todos = np.append(labels_todos, globals()['labels_%s_%s' % (x, y)], axis = 0)
                count += 1
            if count == 4:
                labels_Interacting = labels_todos
            elif count == 5:
                labels_Not_interacting = globals()['labels_%s_%s' % (x, y)]
            elif count > 5:
                labels_Not_interacting = np.append(labels_Not_interacting, globals()['labels_%s_%s' % (x, y)], axis = 0)

full_data_set = datos_Interacting
full_label_set = labels_Interacting

print("##################################")
print("Full_data_set = " + str(full_data_set.shape))
print("Full_label_set = " + str(full_label_set.shape))
print("##################################")

# Separa los datos en training set y test set

X_train_orig, X_test_orig, Y_train_orig, Y_test_orig = train_test_split(full_data_set, full_label_set, test_size=0.10, random_state=42)

X_train = X_train_orig
X_test = X_test_orig
Y_train = Y_train_orig
Y_test = Y_test_orig

print("X_train = " + str(X_train.shape))
print("X_test = " + str(X_test.shape))
print("Y_train = " + str(Y_train.shape))
print("Y_test = " + str(Y_test.shape))
print("##################################")

###### Construccion de la red neuronal est ndar en TensorFlow 2 #####

# Convierte las categorias del training set y del test set en matrices One Hot

from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore', sparse = False)
enc = enc.fit(Y_train)

Y_train = enc.transform(Y_train)
```

```
Y_test = enc.transform(Y_test)

print ("number of training examples = " + str(X_train.shape[0]))
print ("number of test examples = " + str(X_test.shape[0]))
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(Y_train.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(Y_test.shape))
print("##################################")

##### Creacion del modelo  #######################################

model = keras.Sequential()
initializer = keras.initializers.GlorotUniform(seed=1)
model.add(keras.layers.Dense(units = 1500, activation = 'relu',
    kernel_initializer=initializer, input_shape = [X_train.shape[0], X_train.
    shape[1]], kernel_regularizer='l2'))
#model.add(keras.layers.Dropout(rate=0.15))
model.add(keras.layers.Dense(units=600, activation = 'relu', kernel_initializer=
    initializer, kernel_regularizer='l2'))
#model.add(keras.layers.Dropout(rate=0.15))
model.add(keras.layers.Dense(Y_train.shape[1], activation = 'softmax',
    kernel_initializer=initializer))

print("##################################")

model.summary()

print("##################################")

model.compile(
    loss = 'categorical_crossentropy',
    #optimizer = 'adam',
    optimizer = keras.optimizers.Adam(lr=0.00011),
    metrics = ['acc']
    )

history = model.fit(
    X_train, Y_train,
    epochs = 2500,
    batch_size = 32,
    validation_data = (X_test, Y_test),
    shuffle = False
    )

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend();

###### Evaluacion del modelo  ################################

print("#################################################################")

print("Test Set:")

model.evaluate(X_test, Y_test)

print("#################################################################")

Y_pred = model.predict(X_test)
```

```python

# Crea la matriz de confusion

from sklearn.metrics import confusion_matrix

cm_test = confusion_matrix(enc.inverse_transform(Y_test), enc.inverse_transform(
    Y_pred))

print("Confusion Matrix Test:")
print(cm_test)

print("#################################################################")
```

Anexo D. Programa de clasificación utilizando una red neuronal recurrente

```python

import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import csv
import sys

import tensorflow as tf
from tensorflow import keras
from tensorflow.python.framework import ops

####### Preparacion de los datos ################################

np.set_printoptions(threshold=sys.maxsize)

dic1 = ["Interacting", "Not_interacting"]                    # Lista de las categorias principales
dic2 = ["Colleagues", "Couples", "Families", "Friends"]      # Lista de las categorias secundarias

count = 0

# Recorre los archivos creados por el programa de extraccion de datos para guardar las variables

for x in dic1:
    for y in dic2:
        with open("Resultados/%s/%s/%s_%s_data_RNN_3.csv" % (x, y, x, y), "r") as f:   # Abre el archivo
        # with open("Resultados/Interacting/Colleagues/Interacting_Colleagues_data_RNN_3.csv", "r") as f:
            reader = csv.reader(f, delimiter = ',')
            globals()['datos_%s_%s' % (x, y)]= np.array(list(reader)).astype(float)
            if count == 0:
                datos_todos = globals()['datos_%s_%s' % (x, y)]
                count += 1
            else:
                datos_todos = np.append(datos_todos, globals()['datos_%s_%s' % (x, y)], axis = 0)
                count += 1
            if count == 4:
                datos_Interacting = datos_todos
            elif count == 5:
                datos_Not_interacting = globals()['datos_%s_%s' % (x, y)]
            elif count > 5:
                datos_Not_interacting = np.append(datos_Not_interacting, globals()['datos_%s_%s' % (x, y)], axis = 0)

count = 0

# Recorre los archivos creados por el programa de extraccion de datos para guardar las categorias
```

```python

for x in dic1:
    for y in dic2:
        with open("Resultados/%s/%s/%s_%s_data_RNN_Labels.csv" % (x, y, x, y), "r") as f:   # Abre el archivo
        # with open("Resultados/Interacting/Colleagues/Interacting_Colleagues_data_NN_Temporal_Labels.csv", "r") as f:
            reader = csv.reader(f, delimiter = ',')
            globals()['labels_%s_%s' % (x, y)]= np.array(list(reader)).astype(int)
            if count == 0:
                labels_todos = globals()['labels_%s_%s' % (x, y)]
                count += 1
            else:
                labels_todos = np.append(labels_todos, globals()['labels_%s_%s' % (x, y)], axis = 0)
                count += 1
            if count == 4:
                labels_Interacting = labels_todos
            elif count == 5:
                labels_Not_interacting = globals()['labels_%s_%s' % (x, y)]
            elif count > 5:
                labels_Not_interacting = np.append(labels_Not_interacting, globals()['labels_%s_%s' % (x, y)], axis = 0)

full_data_set_orig = datos_Interacting
full_label_set_orig = labels_Interacting

print("##################################")
print("Full_data_set = " + str(full_data_set_orig.shape))
print("Full_label_set = " + str(full_label_set_orig.shape))
print("##################################")

# Crea la matrix 3D de datos para la red recurrente

step = 0
apex = np.array([])
full_label_set_3D = np.array([])

for i in range(full_data_set_orig.shape[0]):
    step = step + 1
    if step == 1:
        apex = np.append(apex, full_data_set_orig[i])

    if step > 1 and step <= 41:
        apex = np.vstack((apex, full_data_set_orig[i]))

    if step == 41 and i == 40:
        full_data_set_3D = apex
        full_label_set_3D = np.append(full_label_set_3D, full_label_set_orig[i])
        apex = np.array([])
        step = 0

    if step == 41 and i > 40:
        full_data_set_3D = np.dstack((full_data_set_3D, apex))
        full_label_set_3D = np.vstack((full_label_set_3D, full_label_set_orig[i]))
        apex = np.array([])
        step = 0

```

```
101 print("Full_data_set_3D = " + str(full_data_set_3D.shape))
102 print("Full_label_set_3D = " + str(full_label_set_3D.shape))
103 print("##################################")
104
105 full_data_set_3D = full_data_set_3D.transpose(2,0,1)
106 print("Full_data_set_3D_Trans = " + str(full_data_set_3D.shape))
107 print("Full_label_set_3D = " + str(full_label_set_3D.shape))
108 print("##################################")
109
110 # Separa los datos en training set y test set
111
112 X_train_orig, X_test_orig, Y_train_orig, Y_test_orig = train_test_split(
        full_data_set_3D, full_label_set_3D, test_size=0.10, random_state=42)
113
114
115 X_train = X_train_orig
116 X_test = X_test_orig
117 Y_train = Y_train_orig
118 Y_test = Y_test_orig
119 classes = np.array([0, 1, 2, 3])
120
121 print("X_train = " + str(X_train.shape))
122 print("X_test = " + str(X_test.shape))
123 print("Y_train = " + str(Y_train.shape))
124 print("Y_test = " + str(Y_test.shape))
125 print("##################################")
126
127 ###### Construccion de la red neuronal recurrente en TensorFlow 2 ########
128
129 # Convierte las categor as del training set y del test set en matrices One Hot
130
131 from sklearn.preprocessing import OneHotEncoder
132
133 enc = OneHotEncoder(handle_unknown='ignore', sparse = False)
134 enc = enc.fit(Y_train)
135
136 Y_train = enc.transform(Y_train)
137 Y_test = enc.transform(Y_test)
138
139 print ("number of training examples = " + str(X_train.shape[0]))
140 print ("number of test examples = " + str(X_test.shape[0]))
141 print ("X_train shape: " + str(X_train.shape))
142 print ("Y_train shape: " + str(Y_train.shape))
143 print ("X_test shape: " + str(X_test.shape))
144 print ("Y_test shape: " + str(Y_test.shape))
145 print("##################################")
146
147
148 ####### Creacion del modelo ##################################
149
150 model = keras.Sequential()
151 initializer = keras.initializers.GlorotUniform(seed=1)
152 model.add(keras.layers.Bidirectional(keras.layers.LSTM(units = 1500,
        kernel_initializer=initializer, input_shape = [X_train.shape[1], X_train.
        shape[2]], kernel_regularizer='l2')))
153 #model.add(keras.layers.Dropout(rate=0.15))
154 model.add(keras.layers.Dense(units=600, activation = 'relu', kernel_initializer=
        initializer, kernel_regularizer='l2'))
155 #model.add(keras.layers.Dropout(rate=0.15))
156 model.add(keras.layers.Dense(Y_train.shape[1], activation = 'softmax'))
```

```python

print("##################################")

#model.summary()

print("##################################")

model.compile(
    loss = 'categorical_crossentropy',
    #optimizer = 'adam',
    optimizer = keras.optimizers.Adam(lr=0.00011),
    metrics = ['acc']
    )

history = model.fit(
    X_train, Y_train,
    epochs = 100,
    batch_size = 32,
    validation_data = (X_test, Y_test),
    shuffle = False
    )

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend();

####### Evaluacion del modelo  #####################################

print("#################################################################")

print("Test Set:")

model.evaluate(X_test, Y_test)

print("#################################################################")

Y_pred = model.predict(X_test)


# Crea la matriz de confusion

from sklearn.metrics import confusion_matrix

cm_test = confusion_matrix(enc.inverse_transform(Y_test), enc.inverse_transform(
    Y_pred))

print("Confusion Matrix Test:")
print(cm_test)

print("#################################################################")
```