UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

CARACTERIZACIÓN DE PATRONES DE ACTIVIDAD MEDIANTE EL SEGUIMIENTO OCULAR

Autor: Radu Dumitru Ghinea

Tutor: Pablo Varona Martínez

Mayo 2016

CARACTERIZACIÓN DE PATRONES DE ACTIVIDAD MEDIANTE EL SEGUIMIENTO OCULAR

AUTOR: Radu Dumitru Ghinea

TUTOR: Pablo Varona Martínez

Grupo de Neurocomputación Biológica

Dpto. de Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Mayo 2016

Resumen

En este Trabajo de Fin de Grado, se ha diseñado y evaluado una metodología que permita reconocer la actividad que realiza una persona con un ordenador a partir de sus patrones de movimiento ocular. La consecución de este objetivo requiere capturar la posición en pantalla de la mirada del usuario mientras ejecuta determinadas tareas.

Esta información puede ser registrada mediante tecnología de seguimiento ocular y los dispositivos que implementan esta tecnología se denominan *eye trackers*. En este proyecto se ha optado por utilizar uno de estos dispositivos: el Eye Tribe Tracker. Se trata de un *eye tracker* de bajo coste, el cual es capaz de medir la posición de las pupilas del usuario y calcular a partir de esta información las coordenadas *x* e *y*, medidas en píxeles, de los puntos en la pantalla.

En primer lugar se han diseñado y programado las herramientas de captura de datos oculares a partir de las librerías proporcionadas por el fabricante del *eye tracker*. El análisis y caracterización de las actividades realizadas por el usuario se ha llevado a cabo por medio de algoritmos de aprendizaje automático. El perceptrón multicapa y el bosque aleatorio son las opciones elegidas para esta tarea.

El estudio llevado a cabo, ha requerido el diseño y ejecución de pruebas para la validación del sistema. En primer lugar, se han seleccionado las actividades a reconocer: la lectura de un texto, el visionado de un video tranquilo, el visionado de un video de acción y la navegación por una página web. Después se ha registrado con el *eye tracker* los patrones oculares de 20 voluntarios durante la realización de estas actividades. Se han reservado datos para entrenar los algoritmos y para evaluar su precisión en la predicción de las actividades. También se ha medido el coste computacional de los algoritmos con el objetivo de evaluar el sistema para su posible uso en tiempo real.

En este documento se recoge la metodología empleada y se analizan los resultados obtenidos en las pruebas de validación. Estos resultados demuestran que es posible reconocer con una precisión suficiente las tareas que realiza un usuario en el ordenador, haciendo uso de un dispositivo de *eye tracking* de bajo coste. Además, es razonable implementar el sistema en un entorno de procesamiento en tiempo real.

Palabras Clave

Seguimiento ocular, The Eye Tribe, Eye Tribe Tracker, perceptrón multicapa, bosque aleatorio.

Abstract

In this undergraduate thesis project, a methodology that allows the recognition of a person's activity with a computer has been assessed based on his/her eye movement patterns. Achieving this aim requires to acquire the user's gaze screen position while executing specific tasks.

This information can be recorded by eye-tracking technology and the devices that implement such technology are called eye trackers. One of these devices has been chosen to be used in this project: the Eye Tribe Tracker. It is a low-cost eye tracker, which is able to measure the user's gaze direction and compute from this information the *x* and *y* coordinates of the screen points, measured in pixels.

First the tools to capture the eye data using the libraries provided by the manufacturer or the eye tracker were designed and programmed. The analysis and characterization of the data was carried out by machine learning algorithms. The multilayer perceptron and a random forest were the options chosen for this task.

The study carried out in this project has required the design and execution of validation tests for the data collected. First, the activities to be recognized have been selected: reading a text, viewing a quiet video, viewing an action video and browsing a web page.

Then, 20 volunteers' eye patterns have been recorded with the eye tracker during the performance of these activities. Data was divided for the algorithms' training and for the evaluation of accuracy in the activity prediction. Also, the algorithms' computational cost was measured in order to evaluate the system for online use.

The methodology used and the results obtained in the tests are reported and analyzed in this document. These results show that it is possible to recognize the tasks performed by a user on a computer with enough precision using a low-cost eye tracking device. In addition, the results also show that it is reasonable to implement the system for an online use.

Keywords

Eye tracking, The Eye Tribe, Eye Tribe Tracker, multilayer perceptron, random forest.

Agradecimientos

Agradezco este trabajo en primer lugar a mi familia, que siempre ha estado a mi lado.

A mis compañeros y amigos, por todos los momentos agradables que hemos pasado juntos en la carrera y fuera de ella.

Y por supuesto a todos los profesores de la carrera, y en especial a Pablo Varona, que con sus conocimientos han ayudado a que este trabajo sea posible.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de ecuaciones

1. Introducción

1.1. Motivación

Desde que aparecieron los primeros ordenadores electrónicos, en la década de 1940, la forma en la cual las personas interactuaban con ellos ha evolucionado enormemente. En aquella época se utilizaban tarjetas perforadas para codificar las instrucciones que la máquina debía ejecutar. Tiempo después se incluiría el teclado como interfaz para el ordenador, permitiendo escribir las órdenes en formato de texto y enviarlas directamente a la máquina (Myers, 1998). Más adelante aparecería el ratón y las interfaces gráficas, mediante los cuales las instrucciones se podían generar mediante la interacción con elementos gráficos mostrados por pantalla. Posteriormente, la llegada de dispositivos con pantallas táctiles ha permitido a las personas interactuar con los mismos a través de gestos dactilares.

No obstante, hoy en día el ratón y el teclado son todavía los dispositivos más utilizados para interactuar con el ordenador. Aunque esta situación puede cambiar en poco tiempo. Desde hace décadas se han desarrollado varias técnicas, además de la tecnología táctil, que pretenden crear nuevos métodos de interacción entre las personas y los ordenadores a partir de señalización biológica. Una de ellas son las interfaces cerebro-ordenador, que tratan de captar las señales eléctricas que se producen en la mente de un usuario para decodificarlas y utilizarlas como comandos (Nicolas-Alonso and Gomez-Gil, 2012). También las tecnologías de reconocimiento de voz, que pretenden reconocer fonética, sintáctica y semánticamente el habla de las personas (Hirschberg and Manning, 2015). Otra técnica es el seguimiento ocular o *eye tracking*, que permite registrar el movimiento de ojos de un usuario mientras observa una determinada escena (Duchowski, 2007; Olsen and Marshall, 2011; Sharafi et al., 2015).

Las primeras investigaciones médicas en temas de actividad cerebral y seguimiento ocular datan de principios del siglo XX. Pero no fue hasta las décadas de 1970 y 1980 respectivamente, cuando se planteó utilizar esta información biométrica para desarrollar interfaces hombre-ordenador.

Las técnicas mencionadas no requieren el uso de extremidades para ser utilizadas. Lo cual abre las puertas no solamente a nuevas formas de dar órdenes a los dispositivos informáticos, sino también a distintos métodos de captar información de los usuarios.

Estas tecnologías pueden utilizarse con diversos fines. Por ejemplo, para mejorar la accesibilidad de los sistemas a personas con discapacidades. También pueden ayudar a detectar problemas cognitivos o sensoriales, como por ejemplo, la dislexia, el déficit de atención, etc. Otros usos, incluyen el análisis de usabilidad con fines de marketing, el control parental o la adaptación automática de interfaces a cada uso.

Por otra parte estas técnicas presentan un problema en general, y es que en muchos casos no es fácilmente accesible económicamente. Al tratarse de tecnología reciente, aún se está desarrollando y perfeccionando. Un sistema de estas características, que proporcione resultados con una gran fiabilidad y precisión tiene un coste elevado. Sin embargo, han aparecido recientemente soluciones de bajo coste que incorporan estas tecnologías. Estos dispositivos sacrifican precisión y fiabilidad en la obtención de datos, con el propósito de ser más asequibles y, por tanto, de llegar a más usuarios.

1.2. Objetivos

El propósito de este trabajo es aplicar la tecnología de seguimiento ocular en un problema de reconocimiento de patrones de actividad, utilizando soluciones de bajo coste. Específicamente, el problema consiste en predecir qué actividad lleva a cabo un usuario frente a un ordenador a partir de los movimientos oculares realizados.

Utilizando un dispositivo de bajo coste se pretende demostrar que es posible alcanzar un nivel de predicción razonablemente bueno, si se aplica un algoritmo adecuado para esta tarea que compense la menor precisión de estos dispositivos.

Para poder resolver con éxito este problema, se han establecido algunos objetivos específicos:

- Determinar el dispositivo a utilizar para realizar el seguimiento ocular a los usuarios. El dispositivo debe ser eficaz y de bajo coste.
- Diseñar un conjunto de tareas que los usuarios deben realizar con el ordenador y reunir una serie de voluntarios para realizar dichas tareas.
- Definir los métodos y algoritmos a utilizar para analizar los datos obtenidos y predecir las actividades a partir de ellos.
- Definir una(s) aplicación(es) que permita(n) comunicarse con el eye tracker, preparar los datos para su análisis y ejecutar los algoritmos de aprendizaje elegidos sobre estos datos, y evaluar dicha(s) aplicación(es) sobre los datos obtenidos.

1.3. Organización de la memoria

El documento elaborado presenta la siguiente estructura:

- **Capítulo 1: Introducción.** En este apartado se realizará una breve introducción, explicando las motivaciones y objetivos del proyecto.
- **Capítulo 2: Estado del Arte.** En este apartado se define el concepto de eye tracking, se explica el funcionamiento general de esta tecnología y se muestran algunos dispositivos que la implementan. También se realiza un repaso breve de los algoritmos de aprendizaje automático utilizados en el proyecto.
- **Capítulo 3: Diseño y Desarrollo.** En esta sección se detallan los aspectos relacionados con el diseño de las pruebas realizadas a los voluntarios, el tratamiento de los datos y los algoritmos utilizados. También se da una breve descripción de la aplicación utilizada en este proyecto.
- **Capítulo 4: Resultados.** En este apartado se exponen los resultados obtenidos de las pruebas con voluntarios y de los algoritmos ejecutados.
- **Capítulo 5: Conclusiones y Trabajo futuro.** En esta sección se exponen las conclusiones que se obtienen de la realización del proyecto y una propuesta de ampliación del mismo.

2. Estado del arte

Para alcanzar los objetivos planteados, es necesario utilizar tecnología específica que sea capaz de obtener información ocular de una persona mientras realiza una determinada tarea en un ordenador. También, se necesita un sistema que analice estos datos y determine la actividad que realiza el usuario. La tecnología que satisface el primer requisito es el denominado *eye tracking* (seguimiento ocular). Para la parte de análisis se han utilizado algoritmos de aprendizaje automático. A continuación se da una explicación general acerca de la tecnología del seguimiento ocular, así como del aprendizaje automático y los métodos usados.

2.1. Introducción Eye Tracking

El concepto de *eye tracking* hace referencia al proceso de medir la actividad ocular de una persona mientras observa una escena, imagen u objeto (Duchowski, 2007). Entre la información que se obtiene de este proceso se encuentra el área o posición en la que una persona concentra la mirada en un momento dado, el tiempo que permanece observando dicha posición, la secuencia de observaciones realizadas y cambios en el diámetro de las pupilas.

La técnica de *eye tracking* se aplica en diversas áreas de estudio y disciplinas (Dimigen et al., 2011; Olsen and Marshall, 2011; Sharafi et al., 2015; Wedel and Pieters, 2007). Destaca su uso en investigación médica, psicología, estudios de marketing y estudios de interacción persona-ordenador. En este último caso, el objetivo es capturar información ocular de un usuario mientras interactúa con la interfaz gráfica de un ordenador.

Dentro de este último contexto, la información obtenida puede utilizarse de dos formas. Tratarla como información de control y utilizar el dispositivo de *eye tracking* como entrada del ordenador. Por ejemplo, para interactuar con el mismo en sustitución del ratón y el teclado (Rozado et al., 2015). O utilizar la información como feedback de la propia interacción del usuario con la interfaz. Por ejemplo, enviando datos sobre el tipo de tarea que realiza un usuario con el ordenador y la forma de realizar dicha tarea. Este es el enfoque que se ha usado en este proyecto.

Como se ha dicho antes, la función principal de los dispositivos de *eye tracking* es determinar donde dirige una persona su visión central. Para lograr este objetivo, los dispositivos modernos utilizan generalmente dos componentes: una fuente de luz infrarroja y una cámara sensible a esta luz. Estos dispositivos se denominan *eye trackers*.

Con la tecnología actual, su funcionamiento se basa en proyectar la luz infrarroja hacia los ojos de la persona, con el objetivo de crear reflejos en la córnea, en la lente ocular y también en la retina. Los tipos de reflejos producidos son cuatro exactamente y se denominan reflejos o imágenes de Purkinje. La primera imagen de Purkinje se conoce como destello (del inglés *glint*) y se produce en la superficie exterior de la córnea. La cámara infrarroja captura este destello, así como la reflexión de la luz en la retina a través de la pupila. De esta forma, el *eye tracker* puede identificar las dimensiones exactas de la pupila (identifica su frontera con el iris), su posición respecto de la cámara y la posición del destello respecto de la pupila. Cuando el ojo se mueve horizontal o verticalmente la posición del destello cambia con respecto a la de la pupila. En la Figura 1 se puede ver este efecto en el ojo de una persona cuando mira directamente, abajo y a

la derecha o por encima de la cámara (siguiendo las representaciones del ojo de izquierda a derecha).

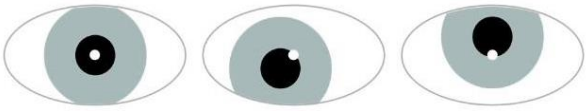


Figura 1. Representación del destello respecto de la pupila en distintas situaciones

De modo que usando el centro de la pupila como referencia se puede trazar un vector entre dicho punto y el destello. Este vector se utiliza para determinar la rotación del ojo respecto a la cámara y después calcular la dirección de la mirada del usuario (medida en grados) y su posición en la pantalla del ordenador.

Según su colocación se distinguen dos variantes de *eye trackers*: los *eye trackers* móviles, los cuales se colocan en la cabeza del usuario, y los *eye trackers* remotos o estáticos, que son los que registran el movimiento ocular a cierta distancia del usuario. Estos últimos están colocados de forma fija próximos al monitor sobre el cual se deseen registrar los movimientos. Por tanto los movimientos del usuario ante la cámara están limitados. En cuanto a los primeros, tienen más precisión y son más adecuados para situaciones en las cuales el usuario debe tener libertad de movimiento con el cuerpo y con la cabeza. No obstante son más intrusivos que los segundos y por tanto más incómodos.

Actualmente, existe una amplia variedad de dispositivos de *eye tracking* de cada tipo y cada uno se caracteriza por ofrecer diferentes características como la precisión del dispositivo, la información que es capaz de capturar y devolver, la frecuencia de captura de esta información y el coste.

A continuación se mencionan algunos de estos dispositivos y sus características más relevantes.

- **Tobii X2-30/X2-60**: Esta es una de las soluciones más destacadas que desarrolla la compañía Tobii Technology (Figura 2). Se trata de un *eye tracker* remoto que ofrece una precisión de hasta 0.4 grados de error en el cálculo de la dirección de la mirada. Obtiene datos de seguimiento como la posición de los ojos respecto a la cámara, la posición de la mirada en pantalla y el diámetro de la pupila. Existen dos versiones del modelo X2, la diferencia entre ambos es la frecuencia de captura de estos datos. La primera versión trabaja a 30 Hz y el segundo a 60 Hz.

Figura 2. Eye tracker tobii X2-30/60

- **RED 250/RED 500**: Es el *eye tracker* desarrollado por SensoMotoric Instruments (Figura 3). Se trata de un *eye tracker* remoto que ya viene integrado en un monitor. Posee una precisión de hasta 0.4 grados y es capaz de obtener

información acerca de la posición de la mirada en la pantalla, movimiento de los ojos y cambios en la pupila. Además es capaz de registrar el contenido presentado por pantalla para análisis posteriores. Existen dos versiones de este dispositivo, una trabaja a una frecuencia de 250 Hz y la segunda a 500 Hz.

Figura 3. Eye tracker RED 250/500

- **Eye Link II**: Se trata de un dispositivo móvil, es decir el montaje se realiza en la cabeza del usuario. Ha sido desarrollado por SR Research (Figura 4). Obtiene la posición en pantalla de la mirada del usuario con precisión media de 0.5 grados de error. Además de este dato, también calcula el diámetro de la pupila. La frecuencia de captura de esta información es de 500 Hz.

Figura 4. Eye tracker Eye Link II

- **Eye Tribe Tracker**: Es un *eye tracker* remoto desarrollado por Eye Tribe (Figura 5). Tiene una precisión entre 0.5 y 1 grado en la dirección de la mirada y puede trabajar a frecuencias de 30 y 60 Hz. Obtiene datos de la posición en pantalla de la visión, diámetro de la pupila y posición de los ojos respecto a la cámara (The Eye Tribe).

Figura 5. Eye tracker Eye Tribe Tracker

Este último dispositivo, el Eye Tribe Tracker, es el que se ha decidido utilizar en este proyecto, principalmente debido a que sus características son suficientes para cumplir los objetivos propuestos. Además, se trata de un dispositivo de bajo coste cuyo precio aproximado es de $100, lo que lo convierte en una opción muy accesible en comparación con el resto de *eye trackers* en los que el precio suele superar los $5000.

2.2. Aprendizaje automático

El aprendizaje automático es una de las ramas de estudio de la inteligencia artificial que tiene como objetivo desarrollar algoritmos que permitan a los programas de ordenador aprender de los datos. Estos algoritmos construyen un modelo computacional a partir de un conjunto inicial de datos de entrada, suministrados en forma de ejemplos. A partir de dicho modelo son capaces de realizar predicciones, reconocer patrones o tomar decisiones sobre los mismos datos u otros nuevos.

Típicamente los ejemplos de entrada están definidos mediante un número fijo de atributos, los cuales pueden ser numéricos o nominales. Dependiendo del tipo de algoritmo los ejemplos pueden tener o no asociados un atributo extra. Este atributo puede tomar valores continuos o discretos, e indica la salida esperada del algoritmo para cada ejemplo.

Según el tipo de aprendizaje, los algoritmos se pueden clasificar en dos grupos:

- **Aprendizaje supervisado**: Estos algoritmos intentan predecir uno o varios atributos de los datos a partir de ejemplos ya etiquetados (incluyen el valor del atributo a predecir). Se caracterizan por construir un modelo computacional que intenta aproximar la relación existente entre el atributo a predecir y los demás (se puede ver como una tarea de aproximar una función). De forma que se puede utilizar posteriormente este modelo para realizar predicciones respecto a este atributo en ejemplos nuevos no etiquetados. Es decir ejemplos en los cuales dicho valor es desconocido.

- **Aprendizaje no supervisado**: El objetivo de esta clase de algoritmos es encontrar relaciones entre ejemplos de entrada no etiquetados atendiendo a su estructura. No se proporciona en los ejemplos ningún atributo extra de clase o numérico que pueda indicar al algoritmo la salida esperada y tampoco corregir el error obtenido. Por ello, estos algoritmos deben buscar regularidades, correlaciones o categorías entre los atributos que describen a los datos para posteriormente poder tomar decisiones acerca de los mismos.

Cabe mencionar que los algoritmos de aprendizaje supervisado dividen típicamente su funcionamiento en dos etapas: entrenamiento/aprendizaje y evaluación/test. Durante la etapa de entrenamiento, se presentan un conjunto de ejemplos de entrada donde cada uno tiene asociado un valor (o varios) correspondiente al atributo (atributos) a predecir, como se ha dicho anteriormente. Dicho valor también indica la salida esperada del algoritmo para cada ejemplo. Con lo cual, al procesar los datos, el algoritmo intenta aproximar su resultado a dicho valor y "memorizar" dicha asociación. Explicado de una forma simple, se puede decir que aprende que salida debe producir para cada entrada. En caso de que el valor de salida del algoritmo difiera del esperado, se comete un error en la predicción. El algoritmo de aprendizaje intenta corregir la información (asociación) aprendida para intentar que las salidas coincidan. En la etapa de evaluación se presentan nuevos datos que no han sido utilizados en la etapa de aprendizaje. El algoritmo produce la salida que mejor corresponde a cada uno de ellos, de acuerdo con la información aprendida.

En este proyecto se han utilizado el perceptrón multicapa y el bosque aleatorio para predecir la actividad que realiza un usuario a partir de los datos de seguimiento ocular. Ambos algoritmos se explican en los siguientes apartados.

2.3. Redes Neuronales Artificiales

Una Red Neuronal Artificial (RNA) es un modelo computacional que intenta imitar el funcionamiento de las redes neuronales de los seres vivos, principalmente su capacidad para reconocer patrones y memorizar información. Las RNA se incluyen como parte de los métodos de aprendizaje automático, concretamente es un método de aprendizaje supervisado. Por tanto, una RNA es capaz de aprender a partir de los datos que se le presentan como entrada y después generalizar lo aprendido a otros datos que no se hayan presentado con anterioridad (López and Fernandez, 2008). Antes de entrar en detalles sobre cómo aprende una RNA, primero se mostrará su estructura y como se procesa la información.

2.3.1. Estructura

2.3.1.1. Capa

La descripción de esta sección se focaliza en las RNA de tipo multicapa. La unidad principal de procesamiento de información en una RNA es la neurona, al igual que en las redes neuronales biológicas. Estas neuronas están conectadas entre sí (no necesariamente todas con todas) formando típicamente una estructura jerárquica de capas. Las capas son agrupaciones de un número variable de neuronas cuyas conexiones entrantes provienen de un origen común y cuyas conexiones salientes se dirigen a un destino común.

A continuación se describen las distintas capas que pueden tener las RNA y en qué orden se transfiere la información entre las mismas:

- Capa de entrada: Las neuronas de entrada obtienen los datos que se van a procesar y los envían directamente a la capa oculta.
- Capa oculta: Las neuronas de esta capa obtienen los datos de las neuronas de la capa de entrada, los procesan y envían los resultados a la capa de salida u otra capa oculta (las RNA pueden tener más de una capa oculta). En esta capa se realiza la mayor parte del procesamiento de los datos.
- Capa de salida: las neuronas de esta capa también procesan la información recibida de la capa oculta (o de entrada, si no tiene capa oculta), obteniendo así el resultado final.

Si una red neuronal artificial consta solamente de capa de entrada y de salida, se denomina red monocapa. Si posee una o más capas ocultas entre la de entrada y la de salida, se denomina red multicapa (López and Fernandez, 2008).

En resumen, los datos originales se proporcionan en la capa de entrada y se envían a través de las conexiones a la primera capa oculta para procesarlos. A partir de aquí los resultados obtenidos en cada capa oculta se envían sucesivamente a la siguiente (en caso de tener más de una capa oculta) hasta obtener un resultado final en la capa de salida. Y en caso de que la red no tenga capas ocultas, los datos originales de la capa de entrada pasan directamente a la capa de salida.

2.3.1.2. Neurona

Como se ha dicho antes, la unidad principal de procesamiento de información en una red neuronal artificial es la neurona artificial. El funcionamiento de la neurona, su estructura y la forma en la que trata la información se detalla a continuación.

La forma en la que la neurona procesa los datos, suele ser igual en las etapas de entrenamiento y evaluación del algoritmo. La única diferencia reside en la modificación de una característica de sus conexiones (el peso) en la primera etapa. Por tanto este detalle se explica en la sección de aprendizaje.

Los datos se envían a la neurona a través de las conexiones que existen entre las mismas. Cada conexión está ponderada, es decir tiene asociado un peso. Después, cada neurona calcula su valor de salida en tres pasos.

Primero aplica una función de propagación (o entrada) para obtener la entrada neta de la neurona. Esta función establece la forma de combinar los valores de entrada y los pesos de las conexiones para calcular un único valor final. La función de propagación más común es la lineal. Consiste en calcular una suma ponderada de los productos entre cada valor de entrada y el peso correspondiente a la conexión por la que este se ha enviado. La Ecuación 2.1 representa este cálculo para una neurona i, siendo w_{ij} el peso de la conexión de la entrada j para esta neurona y x_i el valor de la entrada a la misma (López and Fernandez, 2008; Matich, 2001).

$$input_i = \sum_{j=1}^{n} w_{ij}\, x_j \qquad (2.1)$$

En segundo lugar, evalúa dicho resultado mediante una función de activación o transferencia que determina el estado de activación de la neurona. La función calcula un valor de activación (o estado) a partir de la entrada neta (con un cierto umbral, en caso de necesitarse). Las tres funciones de activación más utilizadas son: la sigmoide binaria, cuyos valores de salida están comprendidos en el rango [0, 1]; la sigmoide bipolar y la tangente hiperbólica, cuyos valores de salida están comprendidos en el rango [-1, 1] (López and Fernandez, 2008; Matich, 2001). Estas funciones están representadas en la Ecuación 2.2, en la Ecuación 2.3 y en la Ecuación 2.4 respectivamente. Los valores mínimos y máximos de los intervalos representan el estado de activación que puede tener una neurona artificial: totalmente inactiva, que corresponde a los valores 0 ó -1, y totalmente activa, que corresponde al valor 1.

$$f(x) = \frac{1}{1+e^{-x}} \qquad (2.2)$$

$$f(x) = \frac{2}{1+e^{-x}} - 1 \qquad (2.3)$$

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad (2.4)$$

El último paso es aplicar la función de salida que determinará la salida de la neurona y, por tanto, la que se proporcionará como entrada a otras neuronas o como salida final de la red. Dependiendo de la función de salida, esta puede tomar un valor de entre dos posibles, comúnmente {0, 1} y {-1, 1}; o tomar cualquier valor dentro de un intervalo, normalmente [0, 1] y [-1, 1] (López and Fernandez, 2008; Matich, 2001). Las funciones de salida más comunes son: no usar ninguna, en cuyo caso la salida es directamente la de la función de activación; y una función binaria. Esta función binaria establece la salida de la neurona a 1 si la función de activación supera un valor de umbral θ y 0 en

caso contrario. También se puede utilizar una función estocástica, cuyo resultado también es binario. En este último caso se utiliza una función de probabilidad de activación y como umbral, un número aleatorio en el rango [0, 1] que se genera siguiendo una distribución de probabilidad uniforme. La salida tomará valor 1 cuando la probabilidad de activación supere dicho valor aleatorio y 0 en caso contrario.

2.3.2. Aprendizaje

El aprendizaje de una RNA se basa en la modificación de los pesos de las conexiones (López and Fernandez, 2008; Matich, 2001). En general, una red neuronal artificial cambia los pesos de acuerdo al error cometido en la salida de la misma. Es decir, una RNA compara los resultados obtenidos con los esperados y, según sea la diferencia entre ambos, autoajusta los pesos de las conexiones para intentar que las salidas coincidan. Uno de los principales algoritmos para las redes multicapa que utiliza el error cometido en la salida para modificar los pesos es el de backpropagation. Se debe mencionar que cuando la etapa de aprendizaje finaliza, estos pesos permanecen fijos y no se pueden modificar en la etapa de evaluación.

2.3.3. Perceptrón multicapa

El perceptrón multicapa es una red multicapa feedforward. Como su propio nombre indica, consta de más de una capa. Es decir, además de incluir la capa de entrada y de salida, tiene una o más capas ocultas intermedias. En el perceptrón multicapa todas las neuronas de una capa están totalmente conectadas con las neuronas de la capa posterior, pero no ocurre así en sentido contrario (las neuronas de una capa no están conectadas con las neuronas de la capa anterior). Como las conexiones son unidireccionales, la información de una capa de neuronas solo se transmite hacia la siguiente. En este caso, las conexiones que establecen las neuronas se denominan conexiones hacia adelante o feedforward (López and Fernandez, 2008). Cabe mencionar también que no solo existen redes feedforward. También existen redes en las cuales la salida de las neuronas de capas posteriores está conectada a las entradas de capas anteriores. Las conexiones que forman se denominan conexiones hacia atrás o feedback (López and Fernandez, 2008).

El perceptrón multicapa basa su aprendizaje en el algoritmo de backpropagation. Su nombre tiene que ver con su funcionamiento. Este algoritmo propaga el error cometido por cada neurona de la capa de salida hacia atrás (hacia las capas de neuronas anteriores). De esta manera es posible modificar los pesos de las conexiones en las capas ocultas.

Cambiar los pesos influye en la entrada neta de las neuronas y por consiguiente en la salida calculada por la función de activación. De modo que, para realizar un ajuste preciso de los pesos y producir la salida adecuada, el algoritmo calcula la variación de la función de activación e incluye este cálculo dentro del proceso de modificación. Esta variación se calcula utilizando la derivada de la función de activación respecto al peso. De esta forma cada peso cambia de forma que la variación del valor de la función de activación (calculada por la función derivada) no produzca un aumento en el error de la capa de salida (o lo que es lo mismo que la diferencia entre la salida real y la salida deseada no se incremente). Dicho de otro modo, el perceptrón multicapa minimiza la función que describe el error cometido en la salida corrigiendo el valor de los pesos y por ende el valor de la función de activación.

La función de error que minimiza el perceptrón multicapa, durante la etapa de entrenamiento, es el error cuadrático medio o ECM. Este se calcula como la suma de los

cuadrados de las diferencias entre los valores de salida de la red neuronal y los valores objetivo de todos los patrones de entrenamiento. Dicho de otra forma, es la distancia cuadrática media entre los vectores de salida y los esperados:

$$ECM = \frac{1}{PN}\sum_{n=1}^{N}\sum_{p=1}^{P}\left(t_{pn} - y_{pn}\right)^2 \qquad (2.5)$$

donde P es el número de patrones predichos, N el número de neuronas de salida, y_{pn} es la salida de la neurona *n* para el patrón *p* y t_{pn} es el valor n-esimo esperado del patrón *p*.

Como se ha dicho, el perceptrón multicapa intenta minimizar la función del ECM a lo largo de varias épocas de entrenamiento. Una época es una ejecución completa del algoritmo de aprendizaje para todos los patrones de entrenamiento. Mediante el error cuadrático medio se puede controlar que el aprendizaje del perceptrón multicapa se realiza correctamente y no se produce sobreajuste o sobreentrenamiento. El sobreaprendizaje o sobreajuste es un efecto que se produce cuando se entrena un clasificador hasta el punto en el que ya no generaliza la información de los patrones de entrenamiento, sino que la memoriza. Es decir, en lugar de reconocer solo ciertas características de los datos y aproximar la salida esperada, intenta retener toda la información (incluso aquella poco relevante o ruidosa) para reproducir con exactitud dicha salida. Esto tiene un efecto negativo para la predicción de patrones nuevos.

En el caso del perceptrón multicapa, cuando se produce sobreaprendizaje, el error del entrenamiento sigue descendiendo o incluso se estabiliza conforme pasan las épocas de entrenamiento, no obstante el error producido en los datos a clasificar aumenta. Esto se debe a que la red ha ajustado los pesos tan bien para aprender los patrones del entrenamiento, que cuando se presenta un patrón no visto no es capaz de predecir la clase correcta a la cual pertenece; a menos que los valores de sus atributos sean prácticamente iguales a los ya vistos.

El sobreajuste de los pesos de la red puede evitarse utilizando alguna de las medidas expuestas a continuación. La más simple es establecer un límite máximo de épocas, de forma que el entrenamiento se detenga antes de que se produzca el efecto descrito. La segunda es establecer un umbral mínimo de variación en el ECM, de manera que si la variación del error es inferior a dicho umbral el aprendizaje se detiene. Otro método es establecer un conjunto de datos de validación a partir del conjunto de entrenamiento original y utilizar este conjunto para estimar el error de clasificación. Después se puede utilizar esta estimación para detener el aprendizaje por ejemplo cuando sea inferior a un cierto valor o cuando comience a aumentar mientras el ECM continúa descendiendo.

2.4. Arboles de decisión

Los arboles de decisión son un modelo computacional que establece un conjunto de reglas para el reconocimiento y predicción de patrones. Su propósito es similar al de la red neuronal artificial y, al igual que estas, están incluidos dentro de los métodos de aprendizaje automático supervisado. Por tanto, el árbol de decisión puede generalizar la información aprendida de un conjunto de datos de entrada a datos no “vistos” durante el entrenamiento.

La descripción de esta sección se centra en especificar la estructura general, el procesamiento de la información y el método de aprendizaje de los árboles de decisión al tratar con valores continuos.

2.4.1. Estructura

2.4.1.1. Nodo

Los árboles de decisión son al fin y al cabo árboles, por tanto son grafos conexos acíclicos y están formados por un nodo raíz, varios nodos intermedios, nodos hoja y ramas (o aristas). Los nodos son los que almacenan información.

Es importante mencionar que la forma en la que el nodo procesa los datos es diferente en las etapas de entrenamiento y evaluación del algoritmo. Por tanto, la forma en la que los nodos tratan la información en la etapa de entrenamiento se explica en la sección de aprendizaje. No obstante, su estructura y la información que almacenan son iguales en ambos casos.

Los nodos internos representan decisiones a tomar con respecto a los datos de entrada. La decisión de un nodo consiste en obtener el valor que toma un atributo en cada uno de los datos y compararlo con un valor constante. Este valor pertenece al rango de valores que puede tomar dicho atributo. Según el resultado de dicha comparación se decidirá el camino a seguir (rama a escoger) para determinar la salida que mejor se aproxime a la esperada para cada dato. Para que este proceso sea posible, cada nodo está asociado a uno de los atributos de estos datos y tienen dos o más ramas que salen de los mismos. Las ramas representan los posibles valores del atributo asociado. En cuanto a los nodos hoja, solamente almacenan la clase más adecuada para cada dato presentado.

A la hora de clasificar un ejemplo, este se presenta como entrada en la raíz del árbol. A partir de aquí, cada nodo realizará la comparación de su atributo asociado: comparará el valor que tiene dicho atributo en el dato y el valor constante almacenado. Según el resultado de la comparación se escogerá una de las ramas del nodo y se pasará el dato al siguiente nodo conectado a dicha rama. Se repite el mismo proceso en cada nodo interno hasta alcanzar un nodo hoja, en el cual no se realizan más comparaciones sino que el árbol devuelve la clase asociada al nodo como la que mejor corresponde al dato.

El operador de comparación más común para atributos continuos es el de menor o igual. La comparación solo puede tener dos resultados, es menor (o igual) o es mayor. Esto produce solo dos resultados para cada nodo, lo que se traduce en tener dos hijos a lo sumo, y el árbol que se genera es binario.

2.4.2. Aprendizaje

La etapa de aprendizaje consiste en crear el árbol de decisión a partir de la raíz y el conjunto completo de los datos de entrada. Durante esta etapa, cada nodo tiene como entrada un subconjunto o partición de los datos de su nodo padre, salvo el nodo raíz como se ha comentado. La forma de obtener estos subconjuntos es partiendo el conjunto de datos del nodo padre por varios puntos. Estos puntos se calculan atendiendo a varios criterios: la calidad de las particiones, el atributo asociado al nodo y la distribución de las clases de salida esperadas en el conjunto de datos.

El aprendizaje de un árbol de decisión se basa en aumentar la calidad de las particiones de datos realizadas en cada nodo. La calidad de una partición se mide por su impureza. La impureza es una métrica que define cómo se distribuyen las clases en el conjunto de datos de un nodo, utilizando las frecuencias de cada clase. El valor más alto de impureza se obtiene cuando las clases están distribuidas uniformemente, es decir todas las frecuencias de clase son iguales. El valor más bajo se obtiene cuando solo existe una única clase en el conjunto de datos.

El objetivo final del aprendizaje de un árbol de decisión es asignar a cada nodo hoja una de las distintas clases, que tendrá que ser la que obtenga el menor número de errores con los datos de entrada. Es decir, se escogerá aquella clase que tenga mayor presencia en dichos datos. Por tanto, lo que trata de conseguir el árbol de decisión es pasar a los nodos hoja subconjuntos de datos en los cuales solo aparezca una clase asociada a todos ellos (intenta conseguir nodos hoja puros). En estos nodos es sencillo asignar la clase de salida, pero no siempre es posible obtener nodos puros. En caso de que aparezca más de una clase, la mejor solución es asignar al nodo la de mayor frecuencia y así obtener una impureza lo más baja posible.

La impureza se calcula de la siguiente manera. Primero se generan los subconjuntos de datos y se calcula la impureza de cada uno de forma individual. Después se calcula la cantidad de impureza total obtenida de la partición a partir de las impurezas individuales. Las medidas de impureza que se suelen utilizar son el índice Gini y el índice de información o entropía (Roche et al., 2009). En este proyecto se ha utilizado el índice de Gini, por ello esta medida se explica en el siguiente subapartado. Para obtener una explicación de la entropía es aconsejable ver el anexo A.

Antes de calcular la impureza de cada subconjunto, se deben generar estos subconjuntos. Para ello, durante el proceso de creación del árbol se debe escoger que atributo se asocia en primer lugar a la raíz y a continuación a los nodos internos. Este atributo se selecciona de entre un subconjunto de los mismos, el cual se puede definir de diversas maneras. Se pueden utilizar directamente todos los atributos de los datos o hacer una selección aleatoria de los mismos. Una vez establecido el subconjunto, se evalúa secuencialmente cada atributo en los datos de entrada de cada nodo. El proceso consiste en ordenar los datos del menor al mayor valor del atributo evaluado actualmente. De esta forma se pueden detectar los posibles puntos en los cuales se puede realizar la partición. Estos puntos serán aquellos en los cuales la clase es diferente entre un dato y el inmediatamente posterior.

El siguiente paso es evaluar cada punto. Se efectúan particiones usando cada uno de los puntos encontrados y se calcula la impureza de los datos como se ha explicado anteriormente, primero individualmente cada subconjunto y luego en promedio. El proceso se repite para todos los puntos encontrados de dicho atributo y para todos los atributos seleccionados. El objetivo es tener una partición con la menor impureza posible, luego la mejor partición será la que tenga más bajo este valor. Una vez encontrada la mejor partición, el nodo memoriza la información que la define, es decir el atributo y el valor por el cual se han dividido los datos. Este valor es la constante usada en las comparaciones con los valores de dicho atributo en los datos de entrenamiento. Además de memorizar la información, se proporcionan los subconjuntos de datos obtenidos de esta partición como entrada a los nodos hijos del nodo procesado.

Llegado a este punto se continúa repitiendo el proceso en los nodos descendientes hasta llegar a los nodos hoja. En estos nodos no se continúa realizando particiones y se guarda la clase mayoritaria como se ha dicho antes. Cuando no se generan nuevos nodos hoja el aprendizaje finaliza.

2.4.2.1. Medida de impureza: Índice Gini

El índice de Gini indica el error medio producido al clasificar un ejemplo cualquiera usando una etiqueta de clase escogida de forma aleatoria, de acuerdo con la distribución

de clases del conjunto de datos. El índice Gini que se obtiene de la clasificación de un conjunto de datos D es

$$G(D) = \sum_{j=1}^{k} p_j (1 - p_j) = \sum_{j=1}^{k} (p_j - p_j^2) = 1 - \sum_{j=1}^{k} p_j^2 \qquad (2.6)$$

donde p_j es la probabilidad de que un ejemplo escogido aleatoriamente pertenezca a la clase j y $(1 - p_j)$ es la probabilidad de que no pertenezca a la clase j (es decir la probabilidad de cometer un error) (Timofeev, 2004). El valor obtenido es el índice Gini de clasificación del conjunto de datos D (G(D)). El índice Gini obtenido de la división del conjunto de datos original en subconjuntos es

$$\sum_{i=1}^{m} p_i \, G(D_i) \qquad (2.7)$$

donde p_i es la probabilidad de que un ejemplo pertenezca al subconjunto D_i. El resultado obtenido mediante la Ecuación 2.7 se utiliza para evaluar las distintas particiones del nodo actual.

2.4.3. Bosque aleatorio

El bosque aleatorio es un método de agregación de clasificadores. Esta técnica consiste en agrupar varios clasificadores con el objetivo de realizar una predicción conjunta sobre los datos. Específicamente, cada clasificador realizará una predicción individual sobre algún atributo (o atributos) de los patrones de entrada (clase para problemas de clasificación o valores para los de regresión). Finalmente, se combinan cada uno de los resultados individuales para generar una única respuesta por cada patrón. Esta combinación puede llevarse a cabo de diversas formas: usando el voto por mayoría de los clasificadores involucrados, mediante un voto ponderado o incluso utilizando un clasificador adicional que decida cómo combinar los resultados.

Concretamente, el bosque aleatorio combina varios árboles de decisión de tipo CART (Classification And Regression Trees) (Bourel, 2012), empleando la técnica de Bagging para determinar la clasificación de los datos de entrada. El Bagging establece cómo combinar las respuestas de los árboles de decisión y que conjunto de datos utilizará cada uno para el entrenamiento. Esta técnica consiste en generar un nuevo conjunto de datos para cada uno de los clasificadores (árboles en este caso) a partir de un conjunto inicial. Es decir que se generan tantos conjuntos como clasificadores hay agregados. Para ello, realiza un muestreo aleatorio con reemplazamiento de los datos originales. Específicamente, se obtienen aleatoriamente datos que se incluyen en un nuevo conjunto con la posibilidad de añadir datos ya incluidos (repetición). Este proceso se repite hasta generar tantas muestras nuevas como árboles se quieran incluir en el bosque aleatorio (Bourel, 2012). Una vez generadas las nuevas muestras de datos se aplica el algoritmo de aprendizaje de los árboles de decisión sobre cada una. Es decir, se crea cada árbol usando su respectiva muestra.

Respecto a dicho algoritmo de entrenamiento, es importante destacar que se utiliza el índice de Gini para medir la impureza de los nodos. Y en cuanto al método usado para generar cada subconjunto de atributos que formará parte del proceso de búsqueda de la mejor partición en cada nodo, se realiza un muestreo aleatorio uniforme. Como consecuencia de incluir tantos procesos aleatorios en la creación de los árboles, resulta muy complicado que dos de ellos utilicen el mismo subconjunto de atributos en todos sus nodos y la misma muestra de datos de entrada. Por tanto, existe una probabilidad muy baja de que los árboles generados sean iguales entre sí. Se debe mencionar también

que los árboles creados son máximos, es decir no se realiza poda de los mismos (Bourel, 2012; Random forests - classification description).

La poda consiste en unificar nodos hoja y varios nodos antecesores a dichos nodos hoja, en un nodo antecesor común. Haciendo esto, se pierde algo de precisión en la clasificación pero se evita el sobreajuste del árbol. Dicho de forma simple, se deshace la partición realizada en todos los nodos sucesores a partir del nodo objetivo de la unificación. O visto de forma inversa, es similar a no seguir realizando particiones a partir de dicho nodo durante el proceso de creación del árbol, convirtiéndose en un nodo hoja.

Como se ha dicho anteriormente, la técnica de Bagging genera muestras con posibilidad de contener datos repetidos. Esto provoca que después de generar una muestra puedan quedar datos del conjunto inicial sin seleccionar y por tanto dichos datos no se utilizan en la construcción del árbol. El bosque aleatorio aprovecha estos datos para crear un conjunto de datos de validación para el árbol correspondiente. Gracias a estos conjuntos de validación se puede aproximar el error de clasificación que generaría el bosque aleatorio en la etapa de evaluación. Este error se denomina out-of-bag error.

La manera de calcular este error se explica a continuación. Una vez construido un árbol de decisión (por ejemplo el k-ésimo), se evalúa cada ejemplo del conjunto de validación de dicho árbol, obteniéndose una clasificación. Al igual que al formar las muestras de entrenamiento mediante Bagging, los ejemplos que forman el conjunto de validación pueden repetirse en otros conjuntos. Es decir, un ejemplo puede quedar fuera de la muestra en varias ocasiones. Al finalizar el proceso de entrenamiento del bosque aleatorio se tiene que cada ejemplo *t* de validación, ha sido clasificado con la clase *c* en la mayoría de los árboles en los cuales dicho ejemplo se ha quedado fuera de la muestra. Por tanto, el número de veces en promedio que la clase *c* no corresponde con la verdadera clase de *t* para todos los ejemplos *t*, es la estimación del error OOB (out-of-bag) (Random forests - classification description).

Hay que tener en cuenta que en los árboles de decisión también se puede generar sobreajuste. Hay que recordar que el árbol de decisión intenta conseguir nodos puros mediante el proceso de entrenamiento explicado anteriormente. Es decir, que intenta obtener nodos hoja en los cuales, los patrones que corresponden a estos nodos son todos de la misma clase o solamente existe un patrón. La única forma de generar nodos totalmente puros es diferenciar perfectamente todos los patrones de entrenamiento, lo que significa que los nodos internos deben contener todas las reglas necesarias para cubrir todos los casos que se presentan. Lo que puede llegar a traducirse en que el árbol de decisión consiga generar al menos una regla para cada atributo que describe el patrón e incluso elaborar varias reglas para un mismo atributo. Si existen atributos poco relevantes en los patrones (es decir que ayudan muy poco a describir las características comunes entre los mismos), estas situaciones aumentan el riesgo de sobreajuste.

El bosque aleatorio permite reducir el riesgo de sobreaprendizaje por la forma en la cual se crean los árboles. En cada nodo no se utilizan todos los atributos posibles en el proceso de selección de la mejor partición, sino un subconjunto aleatorio de los mismos. De esta forma es más complicado que el árbol consiga generar una regla por cada atributo de los patrones.

Para reducir aún más el efecto de sobreajuste existen otros métodos que se aplican en la creación de los árboles de decisión. Uno de ellos es la poda del árbol a posteriori, como

se ha mencionado anteriormente. Lo que intenta conseguir la poda es evitar el sobreaprendizaje de los ejemplos del entrenamiento, a costa de tener menos precisión en la clasificación. El otro método consiste en detener el proceso de creación del árbol antes de que se ajuste perfectamente a todos los ejemplos.

En cuanto a la evaluación del bosque aleatorio, se debe mencionar que se proporcionan los datos a clasificar a cada árbol del mismo. Una vez procesados se obtienen varias salidas, una por cada árbol. Como solo se necesita una única solución, la técnica de Bagging establece que la clase que mejor se adapta a cada dato es la que haya obtenido el "voto" mayoritario entre todos los clasificadores (es decir la salida que más veces ha aparecido) (Bourel, 2012).

3. Diseño y Desarrollo

En la siguiente sección se especifican las características del eye tracker utilizado en las pruebas realizadas con voluntarios, cómo se han diseñado dichas pruebas, la forma de preparar los datos para ser analizados por los algoritmos de aprendizaje y los aspectos de diseño de estos algoritmos. Y también se describe brevemente la aplicación creada bajo estas especificaciones de diseño.

3.1. Diseño

3.1.1. Eye Tribe Tracker

En este proyecto se ha hecho uso del dispositivo Eye Tribe Tracker de la compañía The Eye Tribe. Se trata de un sistema de bajo coste ($100) que incorpora la tecnología de Eye Tracking ya comentada. Este sistema es capaz de calcular en qué lugar de una pantalla (u otro dispositivo que utilice una interfaz gráfica) está mirando una persona a partir de la información extraída de la posición de los ojos.

Con el objetivo de obtener un aprovechamiento óptimo de las capacidades del Eye Tribe Tracker, se han seguido algunas indicaciones de la compañía The Eye Tribe acerca de cómo realizar una correcta configuración y utilización del dispositivo (The Eye Tribe). Algunas de estas indicaciones se exponen en el anexo B.

3.1.1.1. Software

The Eye Tribe también proporciona el software con la funcionalidad necesaria para utilizar el Eye Tracker, además de una API de comunicación con el mismo. Esta API puede ser utilizada en las aplicaciones de terceros que quieran aprovechar las funcionalidades del Eye Tracker en las mismas.

El software sigue una arquitectura de aplicación tipo cliente-servidor y se compone de dos partes: EyeTribe UI y el EyeTribe Server.

El EyeTribe Server es un servidor de aplicaciones. Se encarga de gestionar ciertos parámetros del Eye Tribe Tracker, así como la comunicación entre este y los clientes. El cliente puede ser directamente el EyeTribe UI u otra aplicación que desee hacer uso de esta tecnología. Para que cliente y servidor puedan comunicarse, el EyeTribe Server establece una conexión con la computadora en la que se ejecuta el cliente. Una vez conectados, el cliente y el servidor se envían información mutuamente de acuerdo con el protocolo de comunicaciones.

Este protocolo consiste básicamente en el intercambio de mensajes en formato JSON. A través de estos mensajes, el servidor envía al cliente los datos de la mirada del usuario así como otra información relacionada con la configuración y el estado del Eye Tracker.

En cuanto al EyeTribe UI, se trata de una aplicación cliente que permite acceder a las funciones del Eye Tracker. Consta de una interfaz gráfica sencilla que permite calibrar el dispositivo y ejecutar el proceso de Eye Tracking. Además, permite visualizar los datos obtenidos por el dispositivo “en vivo” en una consola e incluso almacenarlos en un archivo de texto.

3.1.1.2. Calibración

Es importante mencionar que antes de proceder con el seguimiento para un determinado usuario, se debe realizar un proceso de calibración del dispositivo para dicho usuario. La razón principal es que cada persona tiene características oculares únicas, por tanto el Eye Tracker debe detectar y medir estas características con el objetivo de estimar la mirada de los usuarios con mayor precisión.

La calibración se puede realizar utilizando el software EyeTribe UI o a través de la API desde la aplicación cliente. En este proyecto se ha optado por la primera opción, ya que el método de calibración proporcionado por EyeTribe es bastante simple y suficientemente efectivo para el propósito de este proyecto.

Si el Eye Tracker está calibrado correctamente, las coordenadas en pantalla de la mirada del usuario se calculan con un error medio que varía entre 0.5^0 y 1^0 en el ángulo visual respecto del sensor de la cámara. Suponiendo que el usuario este sentado a 60 cm de la pantalla, el error medio correspondiente en pantalla varía entre 0.5 y 1 cm (The Eye Tribe).

Una vez calibrado, el Eye Tracker envía en tiempo real las capturas realizadas durante el seguimiento ocular a una frecuencia regulada por el servidor. El dispositivo de Eye Tribe se puede configurar para capturar datos a velocidades de 30 o 60 Hz (30 o 60 capturas por segundo). Usando una frecuencia alta de captura se obtienen muchos más datos por segundo y, por tanto, más precisión en el seguimiento ocular. Por el contrario, usando una frecuencia alta la cámara es también más sensible a movimientos no oculares del usuario (cabeza y cuerpo) (The Eye Tribe). Por ello, para la obtención de resultados de este trabajo se ha utilizado una frecuencia de 30 Hz, ya que permite a los usuarios más flexibilidad de movimiento (ver anexo B).

3.1.1.3. Datos proporcionados

Cada captura contiene principalmente los datos estimados de la mirada del usuario, junto con otros datos relacionados con el estado y la calidad del seguimiento ocular. Entre los datos más relevantes para la realización de este trabajo, se encuentran las coordenadas de la mirada para un solo ojo y para ambos ojos y el diámetro de la pupila de cada ojo (The Eye Tribe).

Las coordenadas de la mirada se refieren al punto dentro de la pantalla que el usuario está mirando actualmente. Estos puntos se definen usando los ejes *x* e *y* del sistema de coordenadas de dos dimensiones, cuyo origen se encuentra en el punto más alto y a la izquierda de la pantalla. Ambas coordenadas, *x* e *y*, se miden en píxeles de la pantalla. Cabe mencionar que estas coordenadas se calculan para cada ojo de manera individual. Es decir, el Eye Tracker devolverá las coordenadas *x* e *y* para el ojo izquierdo y para el ojo derecho.

El Eye Tracker devuelve también las coordenadas conjuntas de la mirada, que no es más que el resultado de hacer la media entre las coordenadas individuales de ambos ojos. También obtiene el diámetro de cada pupila medido en milímetros.

Para más información acerca del funcionamiento del dispositivo se aconseja consultar el anexo B o la web de The Eye Tribe.

3.1.2. Pruebas con voluntarios

De acuerdo con el objetivo principal del proyecto, se han realizado algunas pruebas sobre distintos usuarios. El propósito de estas pruebas ha sido la adquisición de datos de los movimientos oculares en pantalla de estas personas, mientras realizaban tareas de distinto tipo. Se ha seleccionado un pequeño conjunto de tareas que un usuario realiza comúnmente con el ordenador en un entorno doméstico. Las tareas escogidas son lectura, visionado de videos y navegación por internet. Es importante mencionar que todas las pruebas se han realizado a pantalla completa en un monitor LED de 24 pulgadas. La principal razón, es que la relación de aspecto de los videos, navegador y texto afecten lo menos posible durante la clasificación posterior de los resultados. A continuación, se detalla el contenido de dichas pruebas, las indicaciones dadas a los usuarios en cada una y su duración.

3.1.2.1. Lectura de un texto

Para la lectura se ha seleccionado una noticia publicada en la web. No obstante para no confundir esta tarea con la de navegación, se ha extraído el texto de la página web y se ha almacenado como documento de texto aparte. Además, se ha ajustado el tipo y tamaño de letra para facilitar la lectura del mismo. Se ha pedido a cada usuario que realizara una lectura comprensiva del texto. Es decir, pausada y usando el ritmo normal de lectura de la propia persona, sin forzar una lectura rápida o lenta. De esta manera se consigue captar de forma precisa un número suficiente de movimientos oculares, así como de asegurar que la lectura se adecua al tiempo de captura de datos utilizado en las pruebas; como se explicará en párrafos posteriores.

3.1.2.2. Visionado de videos estático y dinámico

En cuanto al visionado de videos, se han utilizado dos videos de distinta temática y objetivo. El primero es una ponencia universitaria realizada en la Universidad Autónoma de Madrid y está orientado a concienciar sobre un tema determinado a los oyentes. Por tanto, las acciones que tienen lugar en el primer video son escasas y se puede decir que el video es "estático". El segundo video es una competición deportiva y su objetivo es el entretenimiento. Aquí, las acciones que tienen lugar son sorprendentes y abundantes, luego se puede decir que el video es bastante "dinámico". En ambos videos no ha sido necesario dar indicaciones extra.

3.1.2.3. Navegación en página web

Por último, para la navegación se ha escogido la página web de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid. Al igual que en la prueba de lectura, se han dado indicaciones a los voluntarios para realizar una navegación pausada. Se ha pedido a cada uno buscar y abrir la guía docente de una asignatura perteneciente al Master en Investigación e Innovación, concretamente la asignatura de Neuroinformática. Hay que recalcar que se ha escogido esta página por contener numerosos enlaces por los que se debe navegar antes de encontrar la guía docente. Con ello se asegura que la navegación se adecua al tiempo de captura de datos utilizado. Esto último se detalla a continuación.

3.1.2.4. Duración de las pruebas

Respecto a la duración de las pruebas, se ha establecido un límite de tiempo en la captura de datos de 25 segundos para todas ellas (el tiempo comienza a contar desde el momento de inicio de cada una). En algunas pruebas, como en la lectura y navegación, se ha pedido al usuario que finalizase por completo las mismas antes de ejecutar la

siguiente, de forma que utilizase para ello un tiempo superior a 25 segundos. Esto asegura la captura de suficientes datos durante dicho tiempo inicial y que estos correspondan exclusivamente a la actividad principal de la prueba.

Como aclaración a esto último, supongamos que un usuario termina la prueba de lectura o navegación en menos de 25 segundos. Entonces es posible que los últimos datos que se obtengan no correspondan con estas actividades. Esta situación está relacionada con la variabilidad de la velocidad de lectura y navegación entre personas. A pesar de haber dado indicaciones idénticas a los voluntarios a la hora de realizar estas pruebas, las velocidades nunca serán iguales. El tiempo que necesita una persona para realizar una lectura comprensiva o una navegación pausada puede ser mayor o menor que para otra. Las causas son de diversa índole: mayor conocimiento del idioma, mayor conocimiento de la distribución de una página web, deficiencias oculares (hipermetropía, astigmatismo), trastornos lingüísticos (dislexia), discapacidades cognitivas (déficit de atención), etc.

En cuanto a los dos videos, el tiempo depende del ritmo de reproducción, que es idéntico entre ambos y fijo para todos los usuarios. Por tanto, los 25 segundos iniciales son suficientes para capturar los datos necesarios y se puede detener manualmente la reproducción pasado ese tiempo.

3.1.3. Codificación de las actividades en clases

Como se ha dicho anteriormente, los métodos de clasificación elegidos producirán la salida que menor error produzca para cada entrada según la información aprendida previamente. Para ello, primero se deben definir las clases en las cuales se van a dividir los datos. De esta manera se definen las salidas que deben reproducir los métodos de aprendizaje. Dado que se trata de una tarea de clasificación, estas clases se definen mediante valores discretos que codifican las actividades del usuario (apartado 2.2).

La codificación de las clases determinará el número de neuronas de salida que necesita la red neuronal artificial para clasificar los datos. Además, se debe recordar que la respuesta de cada neurona de salida del perceptrón multicapa depende de la salida de una función de transferencia de tipo sigmoidal. Esto quiere decir que esta respuesta no es un valor discreto. Por tanto se deben realizar aproximaciones sobre dichos valores y después establecer una correspondencia con las clases discretas utilizadas. En cuanto a los arboles del bosque aleatorio, las clases se almacenan en los nodos hoja con el mismo formato que en los datos de entrada, es decir valores discretos.

Como se ha comentado en el anterior apartado, las tareas a codificar son cuatro: lectura, visionado de video 1, visionado de video 2 y navegación. Por tanto, se ha asignado a cada tarea una clase distinta y para codificar cada clase se ha utilizado una codificación de clases binaria (para facilitar el cálculo de las clases en la RNA).

Las clases quedan codificadas respectivamente de la siguiente manera: 1000, 0100, 0010, 0001. Los dígitos binarios no tienen significado numérico, simplemente se usa el valor de cada dígito junto con la posición que ocupan en la secuencia de dígitos para determinar la clase.

3.1.4. Series temporales y procesado de datos

Los datos capturados por el Eye Tribe Tracker se envían regularmente en el tiempo al cliente de la aplicación según la frecuencia de captura de la cámara. En este proyecto se

ha usado una frecuencia de 30 Hz, luego se recogen 30 datos por segundo. Estos datos se pueden representar como una serie temporal.

Una serie temporal es precisamente una secuencia de datos, observaciones o valores medidos en determinados momentos del tiempo y ordenados cronológicamente. Estas observaciones se recogen normalmente en instantes de tiempo espaciados entre sí de manera uniforme. En este proyecto las series permiten describir la secuencia de movimientos oculares del usuario a lo largo del tiempo mientras realiza una determinada actividad. Sin embargo, la arquitectura del perceptrón multicapa y del bosque aleatorio, no permiten contemplar directamente la estructura temporal de estas series. Por ello se han transformado los datos originales devueltos (en formato JSON) por el eye tracker, codificando la estructura temporal en una estructura espacial. El resultado son secuencias de posiciones en pantalla ordenadas. Estas secuencias constituyen los ejemplos de entrada de los algoritmos.

3.1.4.1. Procesado de los datos

Primero se han almacenado los datos en un fichero de texto diferente por cada actividad a predecir. Es decir, los datos correspondientes a la actividad de lectura en un fichero, los correspondientes al Video 1 en otro fichero y así sucesivamente. También se han separado estos ficheros según el usuario que realizase la actividad. En cada uno de ellos, cada fila corresponde a una captura de la cámara y la información que esta contiene se reparte en columnas. Los datos utilizados en las pruebas son el tiempo medido en milisegundos en el cual se realizó la captura y las coordenadas *x* e *y* de la posición de los ojos en pantalla. Estos datos forman una serie temporal.

En segundo lugar se han generado patrones ordenados a partir de la serie temporal, utilizando las coordenadas de los puntos de la mirada. Para conseguir el objetivo principal de este proyecto no es necesario conocer el tiempo en el cuál se realizaron las capturas, basta con conocer y respetar el orden en el cual se realizaron. Para obtener los patrones de un determinado fichero, primero se establece un valor que determina el rango de puntos que formarán parte de cada patrón (por ejemplo *n*). El mismo valor será usado en todos los ficheros para asegurar la coherencia de los patrones. Después, se obtienen los *n* primeros puntos (coordenadas *x* e *y* de las n primeras filas) para generar el primer patrón. A continuación, se avanza el rango un punto (una fila) cada vez que se genere un patrón. De manera que cada nuevo patrón tendrá n-1 puntos en común con el anterior. Cada coordenada de cada punto se coloca en el patrón de forma secuencial y en el mismo orden en el cual ha sido obtenido del fichero original (primero la coordenada *x* y después la *y*). Por último, a cada patrón se le añade la clase a la que pertenece al final de la secuencia de coordenadas o no (dependiendo si el patrón es para entrenamiento, para clasificación de ejemplos etiquetados o no etiquetados). Finalmente, estos patrones se almacenan en un nuevo fichero.

Dado que cada punto está definido mediante dos coordenadas x e y, habrá una columna en los patrones por cada coordenada de cada punto. Por tanto, el número de atributos de entrada de los clasificadores será el número de puntos usados en cada patrón de la serie temporal multiplicado por dos. Se han utilizado 15 puntos para elaborar cada patrón temporal, luego en total son 30 atributos de entrada. Esta decisión se explica en los apartados 3.1.5 y 4.2.1. Hay que recordar que se ha añadido la clase al final de estos atributos de entrada y las clases se definen mediante dígitos binarios. Por tanto, el número de atributos de salida de los clasificadores será la cantidad de dígitos utilizados

en la clase, es decir 4. En conclusión, la cantidad total de atributos que describen cada patrón es 34 (30 sin incluir la clase).

Para asegurar la coherencia de los resultados, tanto para el entrenamiento como para la clasificación, se ha fijado el número de puntos totales que forman parte del proceso de creación de los patrones. Esto ha sido necesario, ya que el número de capturas obtenidas para cada actividad de los usuarios no ha sido constante. Lo cual se debe a que el eye tracker detecta algún fallo durante el proceso de seguimiento (por ejemplo, fallo en la detección de ambos ojos). Con esta medida todos los ficheros contienen el mismo número de patrones. El número de puntos de la serie temporal para cada usuario y actividad es de 500, lo que equivale a 17 segundos aproximadamente de grabación en los datos originales. El número total de patrones obtenidos de 20 voluntarios y las 4 actividades son 38880 (486 patrones por serie temporal, por 20 y por 4).

Mediante el proceso expuesto anteriormente, se obtienen ficheros separados según la actividad asociada a los patrones que contienen y el usuario que realizó la actividad. Los ficheros destinados a la etapa de entrenamiento del clasificador se unifican en un único archivo. Si los ficheros se utilizan en la etapa de evaluación, no se unifican sino que se clasifican individualmente.

3.1.4.1.1. Normalización

Es importante aclarar que antes de que los clasificadores puedan trabajar con los datos, es recomendable que estos se encuentren normalizados. El motivo es la heterogeneidad del orden de magnitud de los atributos. Por ejemplo, la coordenada *x* y la coordenada *y* se representan en pixeles y se definen en un rango de valores diferente (de 0 a 1920 y de 0 a 1080 respectivamente). Esto afecta a las operaciones que efectúan los clasificadores sobre los valores de entrada. Los resultados de las mismas estarían determinados en mayor medida por el atributo cuyo rango de valores sea superior, en este caso la coordenada *x*.

Se ha decidido aplicar una normalización a los atributos. La Ecuación 3.1 indica cómo se aplica esta normalización, donde x_i es la entrada del atributo i-ésimo, x_i es la entrada normalizada para el atributo i-ésimo y μ y σ son la media y la desviación estándar de las entradas para dicho atributo.

$$x'_i = \frac{x_i - \mu}{\sigma} \qquad (3.1)$$

3.1.5. Perceptrón multicapa

La red neuronal elegida para clasificar los datos obtenidos mediante eye tracking es un perceptrón multicapa.

Dado que se ha elegido una codificación con cuatro dígitos binarios, la red neuronal necesita 4 neuronas de salida para obtener la clase, una por cada dígito. El valor discreto de la clase se obtiene aproximando la salida de la neurona con el valor más alto a 1, y el resto de valores se aproximan a 0.

En cuanto a la entrada, el número de neuronas en esta capa dependerá del número de puntos utilizados para generar los patrones a partir de las series temporales. Como se ha dicho antes, se han usado 15 puntos para esta tarea. Por tanto, el número de atributos de entrada es 30 y ese es también el número de neuronas que se necesitan en la capa de entrada del perceptrón.

La capa oculta de la red neuronal no necesita ajustarse a ningún número concreto de atributos de entrada o salida. La cantidad de neuronas en esta capa puede escogerse simplemente atendiendo al error cuadrático medio de la RNA (ver sección 4.2.1). De acuerdo con este criterio se ha determinado que una buena opción es utilizar 15 neuronas en la capa oculta.

En cuanto a las funciones de las neuronas, necesarias para el tratamiento de la información, se han escogido la función lineal como función de propagación, la función sigmoidal bipolar como función de activación y ninguna función como función de salida.

Para controlar el sobreaprendizaje de la red, en este proyecto se ha elegido únicamente la opción de establecer un número máximo de épocas (ver apartado 2.3.3). Principalmente, porque los resultados obtenidos tanto en el entrenamiento como en la clasificación de los datos han sido satisfactorios (ver sección 4.2.2) y no se han necesitado medidas adicionales.

En cuanto a las otras opciones, usar un conjunto de validación tiene como contrapartida tener que eliminar algunos patrones del conjunto de entrenamiento. Lo cual implica que la red neuronal tiene menos datos de los cuales extraer información. Y en el caso de que algunos de los patrones eliminados sean característicos de la actividad a la cual corresponden, la red neuronal cometerá un mayor error al intentar predecir estas actividades.

Respecto a establecer un umbral mínimo en la variación del error, la decisión no es sencilla. Se debe escoger el valor adecuado que resuelva el problema del sobreaprendizaje sin que perjudique el entrenamiento de la red neuronal. Puede ocurrir que el aprendizaje se detenga demasiado pronto si se elige un umbral muy grande. Y si se elige un umbral demasiado pequeño, es probable que la variación del error tarde muchas épocas en alcanzar dicho valor y por consiguiente que se genere el sobreajuste.

3.1.6. Bosque aleatorio

Se ha elegido el bosque aleatorio como otro método alternativo para clasificar los datos obtenidos mediante eye tracking.

El número de árboles de decisión que forman el bosque aleatorio se ha elegido de acuerdo con el valor del error out-of-bag (ver sección 4.2.1). Dado que el bosque aleatorio es un conjunto de clasificadores, la predicción es, en general, mejor cuanto mayor sea el número de los mismos. Esto se debe a que se producen más votos para la clasificación final y por tanto hay más posibilidades de que el voto mayoritario sea el correcto, lo que se traduce en un menor error OOB. Teniendo en cuenta lo explicado anteriormente, se ha decidido que una buena opción es utilizar 20 árboles.

Los árboles de decisión que forman el bosque, utilizan el índice de Gini como medida de impureza y el muestreo aleatorio uniforme para generar los subconjuntos de atributos del proceso de búsqueda de la mejor partición en los nodos internos.

Además, los nodos hoja almacenan la clase como caracteres de texto. Por tanto, no afecta a la arquitectura del nodo ni del árbol y estos se pueden adaptar a la codificación elegida para la clase sin problema.

En cuanto a los métodos elegidos para evitar el sobraajuste de los árboles, se ha decidido detener el proceso de creación del árbol de forma anticipada. Este método se suma al método implícito a la construcción de los árboles del bosque explicado en el apartado 2.4.3. No se ha utilizado el método de poda, principalmente porque contradice el diseño del bosque aleatorio. Hay que recordar que los árboles de decisión deben ser máximos y por tanto no se puede aplicar poda a los mismos.

La manera de detener la creación de los árboles de decisión ha sido establecer un valor mínimo de patrones de entrenamiento en los nodos internos, de forma que no se crearán más nodos (particiones) si el número de patrones es inferior a este límite.

El valor que se debe elegir no puede ser demasiado pequeño, ya que no se evitaría el sobreaprendizaje, ni muy alto, ya que el aprendizaje del árbol se detendría demasiado pronto y no obtendría precisión suficiente para clasificar correctamente. Teniendo esto en cuenta, se ha decidido establecer el límite al 10% del número de patrones de entrenamiento, en caso de que este número sea inferior a 10 el límite es directamente 1.

3.2. Desarrollo

3.2.1. Aplicación

Dado que el objetivo principal de este trabajo es reconocer patrones de movimiento ocular, se ha desarrollado una aplicación en Java, utilizado el SDK proporcionado por The Eye Tribe, que implementa las funcionalidades necesarias para tal fin. La elección de usar Java se debe a que es un lenguaje de programación orientado a objetos, lo que permite modular adecuadamente la aplicación y ajustarse mejor a las especificaciones de diseño del apartado anterior. Además permite reutilizar fácilmente el código a la hora de desarrollar aplicaciones basadas en eye tracking para dispositivos móviles Android.

Teniendo en cuenta las especificaciones de diseño explicadas en la sección 3, la aplicación permite obtener los datos del eye tracker, crear los ejemplos de entrada para los algoritmos a partir de ellos y clasificar los patrones aplicando estos algoritmos. Cada una de estas funcionalidades queda implementada por uno de los tres módulos principales que forman la aplicación: módulo de adquisición de datos, módulo de tratamiento de datos y módulo de clasificación de datos.

3.2.1.1. Módulo de adquisición de datos

El módulo de adquisición consta de las clases necesarias para comunicarse con el Eye Tracker, obtener los datos de la mirada del usuario y almacenarlos en los ficheros de texto correspondientes.

Concretamente, dicho módulo se encarga de establecer una conexión con el servidor a través de la API proporcionada por The Eye Tribe. Una vez establecida, comienza a recoger los datos de las capturas que envía el servidor en formato JSON y a una frecuencia de 30 Hz, como ya se ha mencionado.

Se debe recordar que se ha limitado el tiempo de captura de datos por cada actividad a 25 segundos. Para controlar este tiempo se ha utilizado un hilo de ejecución con temporizador. Cuando se inicia el hilo, este obtiene los datos del servidor durante el tiempo indicado y los almacena en el fichero correspondiente usando el formato ya mostrado. Cuando transcurren 25 segundos, el hilo detiene su ejecución y no continúa recibiendo más capturas del servidor. Además de establecer un temporizador para controlar la duración de las pruebas, se ha creado otro a modo de retardo. Dicho retardo

marca el inicio del hilo y, por tanto, de la captura de los datos. Este retardo se ha introducido como margen de preparación para la realización de cada prueba.

3.2.1.2. Módulo de tratamiento de datos

El módulo de tratamiento de datos se encarga de crear los ficheros de patrones a partir de los ficheros de datos originales (apartado 3.1.4). Este módulo permite indicar el número máximo de datos que intervienen en la creación de los patrones, el número de puntos que componen cada uno de los mismos y si los patrones tendrán asociada la clase a la cual pertenecen.

3.2.1.3. Módulo de clasificación de datos

El módulo de clasificación de datos incluye la implementación de todos los clasificadores utilizados en el proyecto, que son la red neuronal y el bosque aleatorio. La ejecución de cada clasificador está dividida en dos partes, entrenamiento y evaluación; dado que implementan algoritmos de aprendizaje supervisado.

Durante la etapa de entrenamiento, el perceptrón multicapa y el bosque aleatorio reciben los patrones de entrenamiento de todas las actividades y usuarios en un único fichero (apartado 3.1.4). En dicho fichero todos los patrones tienen asociada la clase a la cual pertenecen. En la etapa de evaluación, ambos clasificadores reciben los patrones a clasificar en ficheros separados por actividad y usuario.

En cuanto al algoritmo de la red neuronal es posible modificar algunos parámetros como el número de neuronas en la capa oculta y el número de épocas de entrenamiento. Respecto al algoritmo del bosque aleatorio el parámetro que se puede modificar es el número de árboles del bosque.

4. Resultados

Utilizando la aplicación creada se han obtenido los datos de la posición de la mirada de 20 usuarios en total para las cuatro actividades seleccionadas. Después se han creado los patrones de entrada a partir de estos datos. Se han reservado los datos de 16 usuarios (unificados en un único fichero) para la fase de entrenamiento y 4 para la fase de evaluación (en ficheros individuales). A continuación, se han entrenado los clasificadores y se han evaluado con los respectivos ficheros. De forma adicional, se ha medido el tiempo computacional de los procesos de entrenamiento y evaluación de los clasificadores. Los resultados obtenidos se exponen y se explican a continuación.

4.1. Caracterización de los datos adquiridos por el eye tracker

Primero se van a mostrar de forma gráfica los datos originales obtenidos por el *eye tracker* durante la realización de las pruebas. Además se incluyen las capturas realizadas a la pantalla del ordenador por cada actividad, de forma que se puedan comparar las posiciones en las cuales ha fijado el usuario la mirada con la imagen real. En las siguientes imágenes se puede observar un ejemplo de los movimientos oculares que ha seguido uno de los usuarios durante la realización de las 4 actividades elegidas.

Se ha decidido incluir solamente las 4 gráficas del voluntario número 01, ya que las gráficas obtenidas para los 19 restantes son similares a las mostradas en este documento. En las imágenes se representa los puntos de la pantalla en los cuales se ha fijado el usuario mientras realizaba las tareas propuestas. El código de color permite seguir la secuencia de movimientos en el mismo orden en el cual ha sido realizado por el usuario. La secuencia de color va desde el rojo intenso que representa los puntos observados al inicio del proceso de captura, pasa por el azul en los puntos intermedios y llega hasta el negro que marca el final de la captura de datos. El pie de figura identifica el tipo de actividad, la barra de color indica el tiempo transcurrido en milisegundos.

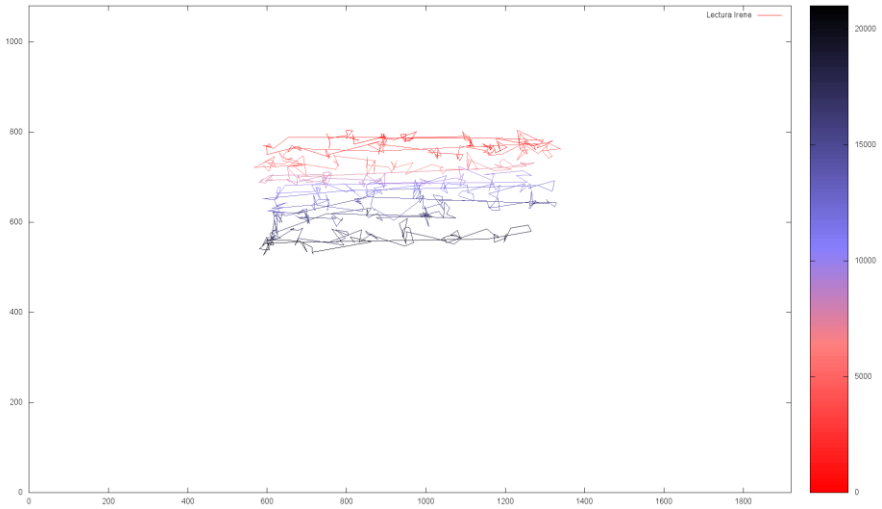
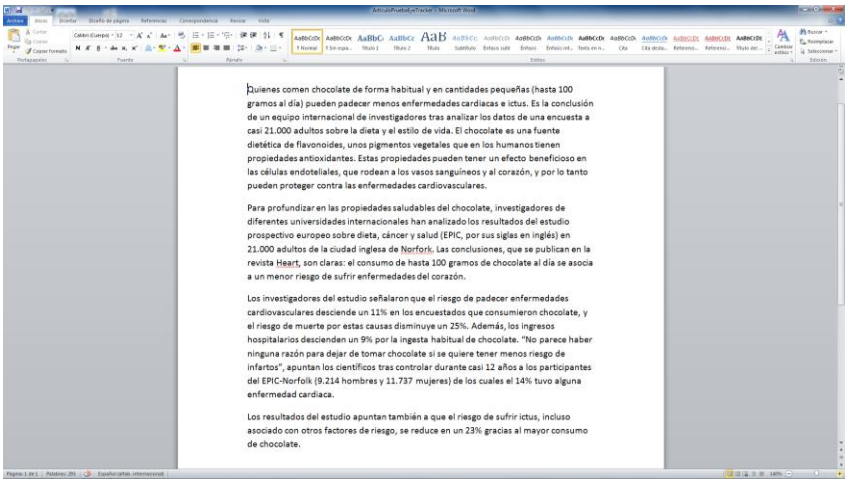


Figura 6. Seguimiento ocular del voluntario 01 durante la actividad de Lectura



Quienes comen chocolate de forma habitual y en cantidades pequeñas (hasta 100 gramos al día) pueden padecer menos enfermedades cardiacas e ictus. Es la conclusión de un equipo internacional de investigadores tras analizar los datos de una encuesta a casi 21.000 adultos sobre la dieta y el estilo de vida. El chocolate es una fuente dietética de flavonoides, unos pigmentos vegetales que en los humanos tienen propiedades antioxidantes. Estas propiedades pueden tener un efecto beneficioso en las células endoteliales, que rodean a los vasos sanguíneos y al corazón, y por lo tanto pueden proteger contra las enfermedades cardiovasculares.

Para profundizar en las propiedades saludables del chocolate, investigadores de diferentes universidades internacionales han analizado los resultados del estudio prospectivo europeo sobre dieta, cáncer y salud (EPIC, por sus siglas en inglés) en 21.000 adultos de la ciudad inglesa de Norfork. Las conclusiones, que se publican en la revista Heart, son claras: el consumo de hasta 100 gramos de chocolate al día se asocia a un menor riesgo de sufrir enfermedades del corazón.

Los investigadores del estudio señalaron que el riesgo de padecer enfermedades cardiovasculares desciende un 11% en los encuestados que consumieron chocolate, y el riesgo de muerte por estas causas disminuye un 25%. Además, los ingresos hospitalarios descienden un 9% por la ingesta habitual de chocolate. "No parece haber ninguna razón para dejar de tomar chocolate si se quiere tener menos riesgo de infartos", apuntan los científicos tras controlar durante casi 12 años a los participantes del EPIC-Norfolk (9.214 hombres y 11.737 mujeres) de los cuales el 14% tuvo alguna enfermedad cardiaca.

Los resultados del estudio apuntan también a que el riesgo de sufrir ictus, incluso asociado con otros factores de riesgo, se reduce en un 23% gracias al mayor consumo de chocolate.

Figura 7. Actividad de Lectura

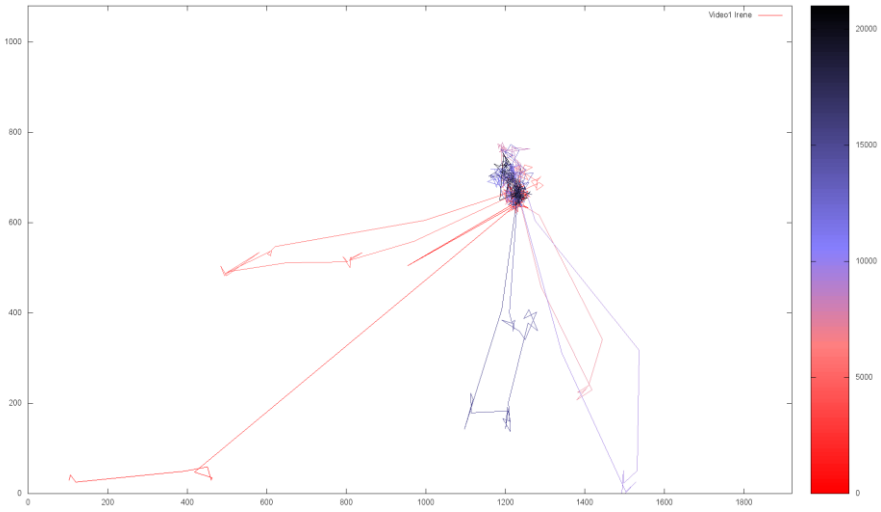


Figura 8. Seguimiento ocular del voluntario 01 durante la actividad de Video 1

Figura 9. Actividad de Video 1

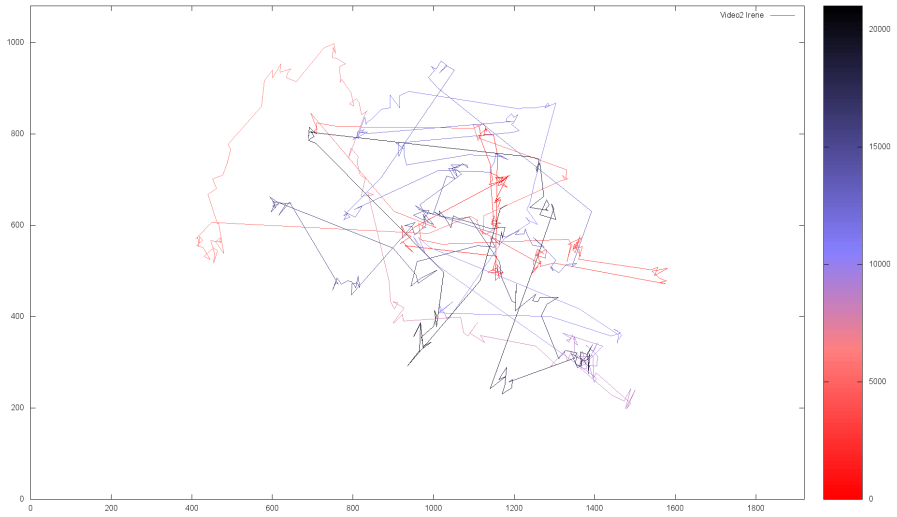


Figura 10. Seguimiento ocular del voluntario 01 durante la actividad de Video 2

Figura 11. Actividad de Video 2

Examinando las dos primeras gráficas y comparándolas con las capturas de pantalla de las respectivas actividades, se puede reconocer con relativa facilidad que efectivamente los movimientos oculares corresponden a la actividad de Lectura y a la del visionado del Video 1 respectivamente. En la primera gráfica se identifican los movimientos de recorrido por las palabras de cada línea y el cambio de línea que el usuario realiza durante la lectura. En la segunda gráfica se observa cómo la gran mayoría de posiciones de la mirada de dicho usuario se concentran en el rostro del protagonista del primer video.

La tercera gráfica pertenece a la actividad del visionado del Video 2. En esta ocasión, no se puede identificar de forma intuitiva la actividad a la que corresponde dicha gráfica por la disposición de los puntos. De hecho, incluso si se compara la gráfica con la imagen de la actividad, no es sencillo identificar las regiones de la escena en las cuales el usuario ha estado fijando su atención.

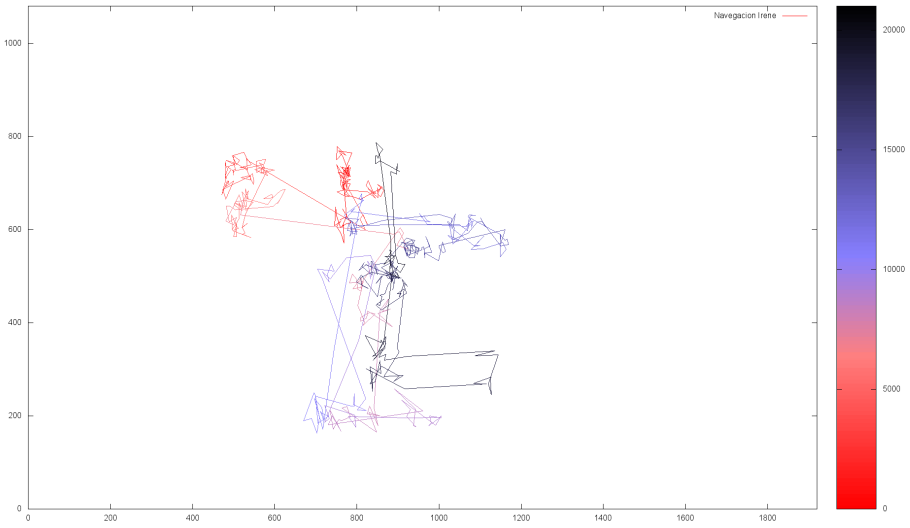


Figura 12. Seguimiento ocular del voluntario 01 durante la actividad de Navegación

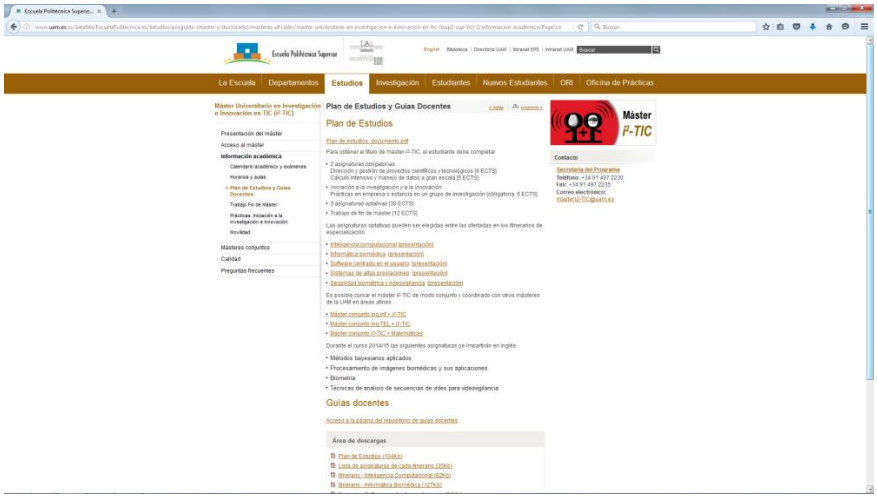


Figura 13. Actividad de Navegación

En la última gráfica se representan los puntos obtenidos en la actividad de Navegación. En este caso también existe cierta dificultad a la hora de reconocer intuitivamente la actividad correspondiente a dicha gráfica. Sin embargo, si se compara con la imagen de esta actividad, se pueden determinar algunas regiones de la página web que el usuario ha recorrido con los ojos. Por ejemplo, los enlaces de la lista de la parte izquierda de la página o los enlaces en la parte central.

Dada la disposición de puntos mostrada en las gráficas anteriores, cabe esperar que la clasificación de los patrones correspondientes a ciertas actividades sea más precisa que para otras. Por ejemplo, se intuye que el perceptrón multicapa y el bosque aleatorio pueden tener más facilidad para reconocer mejor la Lectura y visualización del Video 1, ya que dichas actividades tienen más probabilidad de contener patrones comunes a varios usuarios. En cambio, deberían tener más dificultades para reconocer la actividad de visualización del Video 2, debido a que no se distinguen regiones que contengan potencialmente dicho tipo de patrones. En cuanto a la actividad de Navegación, los resultados de clasificación deberían estar en un punto intermedio entre los de las actividades anteriores (es decir mejor que el Video 2 pero peor que el Video 1 y la Lectura).

4.2. Resultados de entrenamiento y clasificación

A continuación se detallan los resultados obtenidos por los algoritmos de aprendizaje supervisado. Para las pruebas de clasificación, primero se entrena el perceptrón multicapa y el bosque aleatorio utilizando los datos recogidos de 16 usuarios para las cuatro actividades elegidas. Después en la etapa de evaluación se pasan los datos de otros 4 usuarios para evaluar el rendimiento real de los clasificadores para casos no vistos durante el entrenamiento.

4.2.1. Configuración de parámetros

Antes de obtener los resultados de clasificación y predicción, se han realizado pruebas para ambos clasificadores variando algunos de sus parámetros. El objetivo es encontrar los valores adecuados de los mismos que reduzcan el error cuadrático medio y el error OOB, del perceptrón multicapa y el bosque aleatorio respectivamente. Reduciendo los errores respectivos, el error en la clasificación de los patrones también se reduce.

Para estas pruebas se ha utilizado un conjunto reducido de patrones y de puntos por cada patrón. La razón de esto es que basta con obtener una estimación de qué valores de los parámetros son más adecuados de acuerdo con los criterios mencionados. Concretamente, se han utilizado 246 patrones por cada fichero de actividad y 5 puntos por cada patrón (10 atributos de entrada), salvo en la prueba de estimación de este mismo parámetro. Se ha comenzado con las pruebas para estimar los parámetros del perceptrón multicapa. Entre estos parámetros se encuentran la tasa de aprendizaje, el número de neuronas de la capa oculta, el número de épocas de entrenamiento y el número de neuronas de entrada.

En primer lugar se ha estimado el valor de la tasa de aprendizaje. La tasa de aprendizaje determina la magnitud de la variación en los pesos durante el aprendizaje de la red neuronal. Dicho de otra manera controla la velocidad de aprendizaje y la calidad del mismo. En esta prueba se ha ejecutado el entrenamiento durante 200 épocas y se han utilizado 5 neuronas en la capa oculta. En este caso no solo se busca que el error cuadrático medio sea el menor posible, sino que además no genere sobreaprendizaje o sobreajuste en la red neuronal. Se han probado tres valores de la tasa de aprendizaje. En la Figura 14, la Figura 15 y la Figura 16 se muestra la progresión del ECM en cada época y en la Tabla 1 se muestra el menor valor de ECM obtenido en el proceso y en que época se obtuvo.

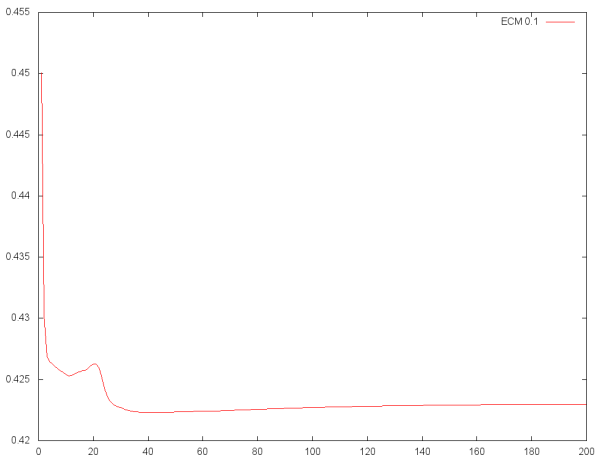


Figura 14. Evolución del Error Cuadrático Medio para tasa de aprendizaje 0.1 y 200 épocas

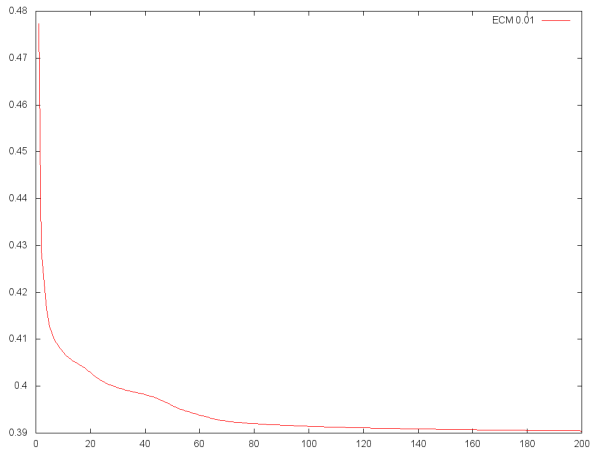


Figura 15. Evolución del Error Cuadrático Medio para tasa de aprendizaje 0.01 y 200 épocas

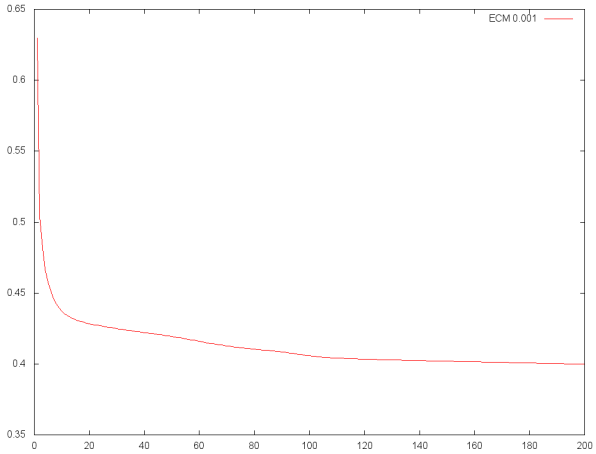


Figura 16. Evolución del Error Cuadrático Medio para tasa de aprendizaje 0.001 y 200 épocas

Tasa aprendizaje	0.1	0.01	0.001
ECM	0.4222	0.3904	0.3998
Época	42	200	200

Tabla 1. Valores de ECM según tasa de aprendizaje

Como se puede observar en las imágenes, el ECM del entrenamiento del perceptrón multicapa cuando se utiliza una tasa de aprendizaje de 0.1, desciende rápidamente en cada época. Esto se debe a que permite variaciones más altas en los pesos de las conexiones. No obstante, también provoca que la red sea menos precisa a la hora de encontrar la configuración de pesos que obtenga el menor ECM global. En otras palabras, la red neuronal no continúa aprendiendo (obtener un menor error con la modificación de pesos) cuando encuentra un mínimo local y a partir de dicho momento el error comienza a aumentar. Esto no es deseable, ya que el objetivo de la red neuronal es encontrar un error mínimo global. Por tanto, la tasa de aprendizaje de 0.1 no es ideal para el perceptrón multicapa, ya que puede provocar que se quede "atrapado" en un mínimo local y también provocar un sobreaprendizaje temprano. Este efecto se puede comprobar en la Figura 14. En la época 42 se obtiene el valor más pequeño del ECM y a

partir de dicho punto comienza a aumentar. Este valor es superior a los obtenidos con el resto de tasas probadas, demostrando que no es un mínimo global.

Con los siguientes valores, 0.01 y 0.001, no se observa que el error se quede "atrapado" en un mínimo local, al menos en el rango de épocas usado. De modo que el ECM no comienza a aumentar de forma prematura. De hecho hasta la época 200 desciende exponencialmente. La diferencia entre la progresión del ECM de los dos valores es que, usando un valor de tasa de aprendizaje 0.001 la magnitud de la variación de pesos es más pequeña que con 0.01. Entonces, la rapidez con la que el ECM alcanzará el mínimo global es más lenta. De hecho en la época 200, el ECM con una tasa 0.001 es mayor que con 0.01. Por ello se ha escogido este último valor que es el más equilibrado en cuanto a estabilidad y rapidez en el aprendizaje.

El siguiente parámetro evaluado ha sido el número de neuronas en la capa oculta del perceptrón multicapa. Para ello se ha utilizado la tasa de aprendizaje escogida anteriormente. La función de las neuronas de la capa oculta es extraer características generales de los patrones en una región particular del espacio de entrada. Dicho de otro modo, transforman la dimensionalidad de los patrones para encontrar características comunes entre los mismos, que no serían fácilmente detectables usando la dimensión original.

Por tanto, el número de neuronas en la capa oculta, afecta a la calidad del aprendizaje y su posterior generalización a ejemplos no vistos. Un número demasiado bajo no podría resolver el problema satisfactoriamente, en cambio un número muy elevado provoca sobreaprendizaje.

Nº Neuronas Ocultas	1	5	10	15	20	25
ECM	0.6100	0.3934	0.3322	0.3003	0.2892	0.2786

Tabla 2. Valores de ECM según número de neuronas en la capa oculta

Se han escogido 15 neuronas para la capa oculta de la red ya que a partir de dicho valor la ganancia de ECM es menor (0.01 de ganancia o inferior), además de que se corre el riesgo de producir sobreaprendizaje.

Otro parámetro es el número de épocas de entrenamiento. Para ello se ha utilizado la tasa de aprendizaje y el número de neuronas ocultas escogidos. En este caso, el número de épocas debe ser suficiente para permitir que la red neuronal alcance el mínimo global del ECM, pero no excesivo por riesgo a que se produzca sobreaprendizaje. No obstante, no es necesario obtener el valor global más pequeño del ECM, ya que dicho valor podría obtenerse solo tras ejecutar un número muy elevado de épocas. Basta con alcanzar un valor suficientemente bajo y próximo al mínimo global. Dado que la progresión del ECM sigue una exponencial negativa, a partir de un cierto número de épocas el error sufre variaciones mínimas. Entonces la diferencia entre el valor mínimo global y otro valor anterior es muy pequeña. En la Tabla 3 se muestra el ECM obtenido después de ejecutar el número de épocas correspondiente. Cada valor corresponde a la última época ejecutada.

Épocas	50	100	250	500	750	1000	2000
ECM	0.3188	0.3108	0.3061	0.3034	0.2988	0.2949	0.2952

Tabla 3. Valores de ECM según número de épocas

Como se puede observar, el error cuadrático medio se reduce a medida que se utilizan más épocas para entrenar el perceptrón multicapa. Es decir, se proporciona más "tiempo" a la red para ajustar sus pesos y esto posibilita que el ECM se aproxime más a su valor mínimo global. En este caso se han escogido 1000 épocas para el entrenamiento de la red neuronal. Dicho valor es suficiente para asegurar que el ECM sea suficientemente pequeño sin llegar a sobreentrenar la red. Un valor superior de épocas puede provocar esta situación, además de que a partir de 1000 épocas el ECM no mejora de forma importante. Por ejemplo, con 2000 épocas el valor obtenido es aproximadamente el mismo que utilizando solo 1000, lo cual indica que el ECM se estabiliza a partir de dicho punto.

Por último se ha hallado el número adecuado de neuronas de entrada que reduzcan considerablemente el ECM sin provocar sobreentrenamiento. En este caso se han generado de nuevo los patrones de los ficheros de actividad variando el número de puntos por patrón. Hay que recordar que el número de neuronas de entrada depende del número de atributos del patrón, y que este valor es el doble que el de puntos por patrón (se utilizan las coordenadas *x* e *y* por cada punto). El número de patrones por fichero de actividad sigue siendo 250 y se aplican los valores escogidos para los parámetros anteriores. Los datos de las pruebas se recogen en la Tabla 4. En dicha tabla se muestra la evolución del ECM del perceptrón multicapa respecto al número de atributos por patrón (teniendo en cuenta ambas coordenadas).

Atributos (neuronas) de entrada	2	10	20	30	40	50
ECM	0.3248	0.2892	0.2429	0.2076	0.1708	0.1562

Tabla 4. Valores de ECM según número de atributos de entrada

Según los valores recogidos en la tabla, salta a la vista que cuanto mayor sea el número de atributos que describen cada patrón más información aprenderá el perceptrón multicapa y por tanto clasificará mejor. Pero también, mayor es el riesgo de sobreaprendizaje, ya que habrá más conexiones que influyan a la hora de calcular la salida de las neuronas y por tanto en el error producido. Por ello se ha escogido el valor de 30 atributos por patrón (15 puntos). A partir de este valor la reducción en el ECM del perceptrón multicapa es menor que respecto a valores anteriores (por ejemplo la mejora en el ECM al cambiar de 20 a 30 atributos es mayor que de 30 a 40).

Una vez configurada la red neuronal, se han realizado las pruebas correspondientes para escoger los parámetros del bosque aleatorio. Estos parámetros han sido el número de árboles en el bosque y el número de atributos de los patrones de entrada. Al igual que con el perceptrón multicapa, se han utilizado 250 patrones por fichero de actividad y 5 puntos por cada patrón, salvo en la prueba de estimación de este mismo parámetro.

Primero se ha estimado el número de árboles del bosque aleatorio que produzca un valor aceptable en el error OOB. Los datos obtenidos se recogen en la Tabla 5.

Nº Árboles	1	5	10	20	50	100
Error OOB	0.2999	0.2902	0.2819	0.2703	0.2667	0.2634

Tabla 5. Valores de error OOB según número de árboles

De la tabla anterior se puede resumir que a mayor número de árboles usados mejor es el error obtenido. Dado que el bosque aleatorio es una agregación de clasificadores y estos clasificadores se entrenan mediante un proceso altamente aleatorio, se obtienen mejores resultados cuanto mayor sea el número de los mismos. Es decir, hay más probabilidad de que se creen árboles que generen una clasificación correcta. Se ha decidido utilizar 20 árboles en el bosque aleatorio ya que el error OOB mejora muy poco para un número de árboles superior.

Finalmente se ha estimado el número de atributos de entrada. En este caso se ha elegido directamente el mismo valor que para el perceptrón multicapa, 30 atributos por patrón, para mantener la coherencia entre resultados. No obstante, para que se pueda apreciar como varía el error OOB con el número de atributos se ha realizado la misma prueba que para la red neuronal. Se han utilizado el número de árboles en el bosque estimado anteriormente.

Atributos de entrada	2	10	20	30	40	50
Error OOB	0.2755	0.2699	0.2589	0.2464	0.2239	0.2114

Tabla 6. Valores de error OOB según número de atributos de entrada

Según los valores recogidos en la tabla, está claro que el error OOB se reduce cuantos más atributos de entrada se utilicen en cada patrón.

A modo de resumen se recopilan los valores de los parámetros que finalmente se han escogido. Para el perceptrón multicapa se ha elegido 0.01 como tasa de aprendizaje, 15 neuronas en la capa oculta, 1000 épocas de entrenamiento y 30 atributos (neuronas) de entrada. Se han elegido 20 árboles para el bosque aleatorio y 30 atributos de entrada. Usando estos valores, el perceptrón multicapa y el bosque aleatorio están configurados de forma adecuada para poder entrenar con el conjunto de datos completo sin generar sobreaprendizaje. Consecuentemente, ambos clasificadores generalizarán correctamente la información aprendida y la clasificación de nuevos datos será precisa.

4.2.2. Clasificación

Una vez establecidos los parámetros adecuados, se evalúa la precisión de los clasificadores sobre los datos de los usuarios reservados para ello. Los resultados de esta clasificación se recogen en la Tabla 7 y en la Tabla 8.

Se debe aclarar que los porcentajes obtenidos se refieren a la proporción de fallos cometidos en la clasificación. Esto es, la cantidad de patrones de un fichero que no corresponden con la clase (y consecuentemente con la actividad) esperada respecto del número total de patrones.

	Lectura	Video1	Video2	Navegación	Media Usuario
Voluntario 17	15.23%	19.14%	31.28%	7.41%	18.26%
Voluntario 18	20.58%	16.46%	22.25%	24.90%	21.04%
Voluntario 19	7.61%	0.82%	25.72%	15.43%	12.39%
Voluntario 20	8.23%	8.64%	28.81%	14.20%	14.97%
Media Actividad	12.91%	11.26%	27.01%	15.48%	

Tabla 7. Resultados de clasificación del perceptrón multicapa (porcentajes de proporción de fallos)

	Lectura	Video1	Video2	Navegación	Media Usuario
Voluntario 17	15.64%	47.74%	41.15%	9.47%	28.5%
Voluntario 18	20.16%	31.07%	14.90%	13.79%	19.98%
Voluntario 19	10.49%	15.23%	30.04%	7.61%	15.84%
Voluntario 20	20.16%	23.87%	30.25%	17.28%	22.89%
Media Actividad	16.61%	29.47%	29.08%	12.03%	

Tabla 8. Resultados de clasificación del bosque aleatorio (porcentajes de proporción de fallos)

Para tener una idea de cuál sería una proporción suficientemente buena de fallos en la clasificación y poder comparar los porcentajes obtenidos, se establecerá a continuación una cota máxima para la tasa de fallos. Para hallar esta cota, se debe recordar que el objetivo principal de los clasificadores usados es asignar la clase correcta a cada patrón de entrada contenido en los ficheros de actividad. Después cada fichero de actividad obtendrá una clasificación final, que será la clase de los patrones predicha con mayor frecuencia.

Es lógico que para que la clasificación sea correcta la proporción de la clase correcta debe ser superior a la proporción individual de cada clase incorrecta. Por ejemplo, si la clase correcta es A, su frecuencia debe ser superior a las frecuencias por separado de B, de C y de D; en un caso en el que aparecen las 4 clases en la clasificación de un fichero. Si la frecuencia de la clase correcta es también superior a la suma de las proporciones de todas las clases incorrectas la condición anterior siempre se cumple.

La tasa de aciertos mínima para la clase correcta que permite asegurar ambas condiciones es el 50%. En el caso de la tasa de fallos esta sería su cota máxima. Por tanto ya se ha encontrado una cota máxima para la tasa de fallos en la clasificación de los ficheros. En caso de que los resultados superen dicha cota, se considerará que la clasificación es poco precisa y fiable. En una aplicación online de reconocimiento de

actividad se podrían utilizar criterios de acierto consecutivos en el tiempo para decidir sobre la actividad con precisión adicional.

Fijándose en los resultados obtenidos se extraen algunas conclusiones. En primer lugar, que la actividad de ver el Video 2 es la que obtiene, de media, la proporción más alta de fallos para la red neuronal, siendo también alta en el clasificador de bosque aleatorio, en comparación con el resto de actividades. Es lo que cabría esperar, ya que como se vio en el apartado 4.1 el segundo video es el que más variabilidad de movimientos oculares posee. Sin embargo, aun teniendo la tasa media de fallos más alta (en algún usuario concreto puede ser ligeramente superior, pero sin superar el 50%), esta es menor al 30%. Este resultado basta para asegurar que ambos clasificadores predicen correctamente la clase final de los datos recogidos para esta actividad. Aunque cabe destacar que el perceptrón multicapa obtiene un resultado ligeramente inferior que el bosque aleatorio, con un diferencia aproximada del 2% entre ambos porcentajes para esta actividad.

En el caso de la Lectura y la Navegación los porcentajes medios de aciertos son bajos en ambos clasificadores, inferiores al 20%. Esto indica que estas actividades tienen menos variabilidad en las coordenadas *x* e *y* respecto al Video 2. Como consecuencia, favorece el aprendizaje de características comunes entre distintos usuarios y el posterior reconocimiento de las mismas. En el caso de la Lectura este resultado coincide con lo esperado intuitivamente tras observar la gráfica del apartado 4.1, aunque solamente por parte del perceptrón multicapa. Para el bosque aleatorio se esperaba que el porcentaje de Lectura fuera inferior al de Navegación. En cuanto a esta última actividad, coincide con lo explicado en el mismo apartado. Pero al igual que para la Lectura, solo en el caso del perceptrón multicapa.

Comparando ambos clasificadores se observa que el perceptrón obtiene de media en la actividad de Lectura mejores resultados que el bosque, aunque ocurre justo lo contrario en el caso de la Navegación. Existe una diferencia de casi 4% entre ambos resultados en la Lectura y aproximadamente un 3,5% en la Navegación.

Por último, los resultados obtenidos en la actividad del Video 1 son buenos para los dos clasificadores. Sin embargo, hay una mayor diferencia entre ambos porcentajes. En el caso del bosque aleatorio la tasa de fallos está cerca del 30% y el perceptrón multicapa aproximadamente 11%. Según la gráfica vista en el apartado 4.1, la mayoría de las coordenadas se concentran en una región muy específica lo que puede facilitar el aprendizaje y clasificación de los patrones. Este hecho queda reflejado en el porcentaje de error de la red neuronal, siendo este el menor de todos ellos. Pero los resultados obtenidos por el bosque aleatorio indican que dicha distribución de los puntos no ha ayudado sustancialmente a que el error sea menor, siendo muy parecido al del Video 2. Incluso en el caso concreto del voluntario 17, el error se aproxima a la cota del 50% por lo que los resultados están próximos de ser poco fiables.

A la vista de estos resultados, se puede concluir que el perceptrón multicapa clasifica mejor las actividades de Lectura, visionado del Video 1 y visionado del Video 2 que el bosque aleatorio. Pero no es así para la actividad de Navegación. Aun así, la diferencia entre los porcentajes de fallos para la Navegación no es muy importante, pero si lo es la diferencia en los porcentajes para el Video 1. Entonces se puede afirmar que generalmente el perceptrón multicapa es más preciso que el bosque aleatorio, en cuanto a la clasificación de los patrones extraídos de las secuencias temporales de *eye tracking* con la metodología empleada.

4.2.3. Coste computacional

Se ha efectuado un análisis adicional en el proyecto para medir el tiempo de ejecución de los clasificadores durante el entrenamiento y la evaluación. El objetivo es evaluar los algoritmos según su coste computacional y valorar si es razonable utilizarlos en un entorno que requiera el procesamiento en tiempo real de los datos de seguimiento ocular.

Los tiempos obtenidos se recogen en la Tabla 9. Es importante destacar que se ha ejecutado cada algoritmo con los mismos parámetros especificados para las pruebas de clasificación del apartado 4.2.2.

	Perceptrón Multicapa	Bosque Aleatorio
Entrenamiento	505133.5 ms (1000 épocas)	2029186.0 ms (20 árboles)
Evaluación	1.6 ms	11.4 ms (20 árboles)

Tabla 9. Tiempos de ejecución en milisegundos

Los resultados mostrados de la Tabla 9 se han obtenido utilizando como entrada el fichero de todos los patrones en la etapa de entrenamiento y solamente un fichero de actividad para la etapa de evaluación. En este último caso, se ha tomado esta decisión dado que cada fichero contiene el mismo número de patrones y de atributos. Luego el coste de la evaluación no difiere sustancialmente entre estos ficheros.

A la vista de los dos primeros resultados, parece que el perceptrón multicapa tiene un menor coste computacional que el bosque aleatorio en ambas etapas. En el entrenamiento, hay que tener en cuenta que el bosque aleatorio es una combinación de clasificadores y debe ejecutar el proceso de aprendizaje para todos los árboles a crear, en este caso 20. En cambio, solo se crea un único perceptrón multicapa que durante el aprendizaje modifica los pesos de las conexiones durante varias épocas de entrenamiento, en este caso 1000. Luego a priori, es más costoso crear un árbol de decisión que modificar los pesos de las conexiones durante una época en la red neuronal. En la etapa de evaluación, en el caso de la red neuronal las épocas no afectan al tiempo de ejecución de la evaluación. En cambio, en el bosque aleatorio sí que afecta el número de árboles de decisión, ya que los patrones de entrada se evalúan en todos ellos. En este caso es lógico que el coste del bosque aleatorio sea superior al de la red neuronal. Aun así, no existe una diferencia excesiva entre ambos resultados: 9.8 ms.

Para comparar en condiciones más similares la ejecución de los algoritmos, se ha realizado una prueba más para cada uno. Se ha entrenado el perceptrón usando una sola época y el bosque usando un solo árbol. Por parte de la evaluación se ha procesado el mismo fichero únicamente para el bosque.

Los resultados obtenidos en este caso en la etapa de entrenamiento han sido 1498.4 ms para el perceptrón multicapa y 131354.6 ms para el bosque aleatorio. Como se puede ver, el coste del perceptrón multicapa sigue siendo menor que el coste del bosque aleatorio. Esto se debe a que la creación de un árbol implica buscar la mejor partición por cada uno de sus nodos internos. Es decir, se calcula el índice de Gini por cada atributo del subconjunto seleccionado del nodo interno y cada posible punto de partición hallado en los valores de estos atributos. Por parte del perceptrón multicapa, el proceso más costoso es calcular la salida de la red, hallar los incrementos de pesos según el error cometido y actualizarlos. Por tanto el perceptrón solo realiza cálculos

aritméticos, no realiza ninguna búsqueda sobre los ejemplos de entrada ni efectúa particiones sobre los mismos y el procesamiento de la información es más rápido. En cuanto a la etapa de evaluación, los nuevos valores obtenidos han sido 1.6 ms para el perceptrón y 1.2 ms para el bosque. Según estos resultados, el bosque aleatorio tiene aproximadamente el mismo coste que el perceptrón multicapa. En este caso se debe a que el perceptrón solo debe realizar los cálculos computacionales para obtener la salida de cada ejemplo de entrada. El árbol de decisión solo realiza comparaciones de valores hasta alcanzar un nodo hoja y devuelve directamente la clase almacenada en el mismo. Luego clasificar todos los patrones de actividad de un fichero requiere un coste computacional muy bajo en ambos clasificadores.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

En este trabajo de fin de grado se ha propuesto realizar el reconocimiento de la actividad que lleva a cabo un usuario frente al ordenador, utilizando para ello un dispositivo de seguimiento ocular de bajo coste: el Eye Tribe Tracker, de la compañía The Eye Tribe. Al tratarse de un dispositivo de bajo coste, tiene unas características limitadas en cuanto a precisión, frecuencia de trabajo y fiabilidad en la obtención de datos respecto a otras soluciones más costosas. Se ha conseguido compensar estas limitaciones realizando un tratamiento adecuado de los datos de seguimiento ocular y utilizando algoritmos de aprendizaje automático que permiten reconocer patrones de actividad a partir de ellos.

Para asegurar el correcto tratamiento y almacenamiento de la información se ha desarrollado un driver, usando el SDK proporcionado por The Eye Tribe, que se comunica con el eye tracker y obtiene directamente los datos del dispositivo. También se han implementado un conjunto de procedimientos que modifican el formato de estos datos para adaptarse a las características de los algoritmos de aprendizaje automático y mejorar sus resultados.

Para asegurar que el aprendizaje y clasificación de estos algoritmos son óptimos, se han ajustado los parámetros correspondientes antes de procesar los datos. Los dos algoritmos de aprendizaje empleados en el proyecto, el perceptrón multicapa y el bosque aleatorio, han obtenido resultados satisfactorios a la hora de clasificar los patrones de movimiento ocular capturados por el eye tracker para un conjunto de actividades específicas. Estas actividades son leer un texto, ver un video tranquilo, ver un video de acción y navegar en un sitio web.

Analizando detenidamente estos resultados, se ha observado que el perceptrón multicapa obtiene generalmente mejores resultados para la parametrización escogida en ambos algoritmos; siendo capaz de reconocer estas actividades con una alta precisión y un bajo coste computacional. Esto no quiere decir que el bosque aleatorio no sea un algoritmo adecuado para la tarea propuesta en este trabajo. De hecho reconoce muy bien los patrones de la actividad de Navegación, obteniendo resultados razonables en las tareas restantes. Y aunque el tiempo de ejecución empleado por este clasificador es superior al perceptrón multicapa, no supone una diferencia excesiva.

Teniendo todo esto en cuenta se concluye que utilizando un eye tracker de bajo coste, es posible reconocer la actividad que realiza una persona delante del ordenador entre un conjunto prefijado de tareas; obteniendo resultados razonablemente buenos en cuanto a acierto en la predicción y coste computacional.

5.2. Trabajo futuro

Como ya se ha expuesto, en el presente trabajo, se ha aplicado la tecnología de eye tracking para capturar los movimientos oculares de un usuario mientras realiza determinadas actividades frente al ordenador. Después se ha utilizado esta información para predecir la actividad realizada mediante algoritmos de aprendizaje automático.

Con vistas al futuro, se pretende ampliar el conjunto de actividades estudiadas en este proyecto. El propósito es aumentar el número de aplicaciones para el reconocimiento de

patrones oculares. Por ejemplo, si se distinguen entre diferentes tipos de videos, como una película o un videoclip, es posible adaptar la interfaz al mismo cambiando la relación de aspecto, la iluminación, el volumen del sonido, etc.

Otra ampliación interesante, sería aumentar el número de puntos utilizados para crear los patrones, y realizar una búsqueda de la parametrización óptima más exhaustiva. Esto permite comprobar si la clasificación de los algoritmos mejora al aumentar la estructura temporal de la entrada. Se ha pensado además en proporcionar como entrada la progresión del diámetro de las pupilas de los usuarios y estudiar si la nueva información introducida mejora la clasificación.

Por otro lado, en este proyecto se han utilizado dos algoritmos de aprendizaje que no tienen en cuenta la estructura temporal de los datos (series temporales). Existen otros algoritmos de clasificación que sí tienen en cuenta esta característica, como es el caso del SMTS (Baydogan and Runger, 2015). Se ha pensado aplicar este tipo de algoritmos al reconocimiento de los patrones de actividad y comprobar si se obtiene más precisión en la predicción de la actividad incluyendo esta información.

Como trabajo extra, se ha planteado extender el reconocimiento de actividades a partir del seguimiento ocular en otro tipo de dispositivos (no exclusivamente en el ordenador). Por ejemplo, en televisores, smartphones, tablets, pantallas de cine, etc (Rozado et al., 2015). Esto requiere efectuar el seguimiento ocular sobre dichos elementos. En algunos de ellos, esta tarea es bastante simple utilizando alguno de los eye trackers de bajo coste disponibles en el mercado (televisores y algunas tabletas). Pero en otros casos resulta más complicado, bien porque su tamaño no se adapta bien al tamaño de pantalla del dispositivo (smartphones y ciertas tablets) o bien, porque se requeriría una gran número de los mismos y una buena coordinación para aplicar esta tecnología (pantallas de cine). Por ello, se planea aplicar el seguimiento ocular en Smart TVs y Tablets, a la espera de que surjan nuevos eye trackers adaptados para Smartphones y pantallas de cine. Y dado que la aplicación creada para este proyecto se ha desarrollado en Java, esta es fácilmente adaptable a todos los dispositivos basados en Android.

También es interesante utilizar una Interfaz Cerebro-Ordenador conjuntamente con un eye tracker, ya que puede mejorar la precisión a la hora de diferenciar ciertos patrones (Dimigen et al., 2011). Por ejemplo las señales de EEG son diferentes al leer un texto que al observar un video. Estos dispositivos también se utilizarían en otro tipo de aplicaciones, como la identificación biométrica de usuarios o como método de entrada de un ordenador mediante el cual enviar instrucciones.

Finalmente, queda como trabajo futuro analizar los datos de seguimiento ocular y predecir la actividad que realiza un usuario en tiempo real además de permitir identificar comandos realizados con gestos oculares (Rozado et al., 2012). Este objetivo requiere emplear algoritmos que sean capaces de procesar los datos de entrada en el menor tiempo posible. Una opción es utilizar algoritmos de aprendizaje automático, como el perceptrón multicapa o el bosque aleatorio, para esta tarea; dado que su tiempo de ejecución es razonable como ya se ha comprobado en este trabajo de fin de grado.

Referencias

Baydogan, M. G., and Runger, G. (2015). Learning a symbolic representation for multivariate time series classification. *Data Min. Knowl. Discov.* 29, 400–422. doi:10.1007/s10618-014-0349-y.

Bourel, M. (2012). Métodos de agregación de modelos y aplicaciones. *Mem. Trab. Difus. Cient. y Tec.*, 19–32.

Dimigen, O., Sommer, W., Hohlfeld, A., Jacobs, A. M., and Kliegl, R. (2011). Coregistration of eye movements and EEG in natural reading: Analyses and review. *J. Exp. Psychol. Gen.* 140, 552–572. doi:10.1037/a0023885.

Duchowski, A. (2007). *Eye tracking methodology: Theory and practice*. Springer London.

Hirschberg, J., and Manning, C. D. (2015). Advances in natural language processing. *Science (80-.).* 349, 261–266.

López, R. F., and Fernandez, J. M. F. (2008). *Las redes neuronales artificiales*. Netbiblo.

Matich, D. J. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. Universidad Tecnológica Nacional.

Myers, B. A. (1998). A Brief History of Human Computer Interaction Technology. *ACM Interact.* 5, 44–54. doi:http://doi.acm.org/10.1145/274430.274436.

Nicolas-Alonso, L. F., and Gomez-Gil, J. (2012). Brain Computer Interfaces, a Review. *Sensors* 12, 1211–1279. doi:10.3390/s120201211.

Olsen, A., and Marshall, P. (2011). Using Eye Tracking for Interaction. *Hum. Factors*, 741–744. doi:10.1145/1979742.1979541.

Random forests - classification description Available at: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#features.

Roche, A., Ghattas, B., Fraiman, R., Grampin, E., Mordecki, E., and Perera, G. (2009). Árboles de decisión y Series de tiempo. Universidad De La Republica Uruguay.

Rozado, D., Moreno, T., San Agustin, J., Rodriguez, F. B., and Varona, P. (2015). Controlling a Smartphone Using Gaze Gestures as the Input Mechanism. *Human–Computer Interact.* 30, 34–63.

Rozado, D., Rodriguez, F. B., and Varona, P. (2012). Low cost remote gaze gesture recognition in real time. *Appl. Soft Comput.* 12, 2072–2084.

Sharafi, Z., Soh, Z., and Gueheneuc, Y. G. (2015). A systematic literature review on the usage of eye-tracking in software engineering. *Inf. Softw. Technol.* 67, 79–107.

The Eye Tribe Available at: https://theeyetribe.com/.

Timofeev, R. (2004). Classification and regression trees (cart) theory and applications. Humboldt University, Berlin.

Wedel, M., and Pieters, R. (2007). "A review of eye-tracking research in marketing," in *Review of Marketing Research*, ed. N. K. Malhotra (M.E. Sharpe Inc.), 123–147.

Glosario

Android: Sistema operativo basado en Linux y orientado a los dispositivos móviles.

API: (del inglés Application Programing Interface) es un conjunto de procedimientos y funciones de una biblioteca que establecen la forma que tiene de acceder el software de terceros a la funcionalidad de la misma.

EEG: Electroencefalograma.

JSON: (acrónimo del ingés JavaScript Object Notation) se trata de un formato estándar utilizado en los sistemas informáticos de arquitectura tipo cliente-servidor, para el intercambio de información. Los datos se transmiten como texto usando una representación de pares atributo-valor.

RNA: siglas de Red Neuronal Artificial.

Smartphone: es un teléfono móvil (denominado inteligente) que posee características y funcionalidades similares a las de un ordenador personal portátil, además de incluir las de un teléfono móvil convencional. Se caracterizan por utilizar la pantalla táctil que incorporan como método de entrada.

Tablet: se trata de un dispositivo móvil de similares características y funciones a las de un ordenador personal portátil, que incluye generalmente pantallas táctiles utilizadas como método de entrada. En este caso, esta clase de dispositivos no incorpora funciones de telefonía móvil.

Anexos

A. Índice de información o entropía

En el ámbito de la teoría de la información, un sistema de comunicación se modeliza como una fuente que genera mensajes y los transmite a un receptor a través de un canal de comunicación. Los mensajes contienen información. En este contexto, la información se trata desde un punto de vista estadístico. Es decir, se relaciona con la libertad de elección que tiene el emisor para seleccionar un mensaje concreto de entre un conjunto de posibles mensajes. Los mensajes están formados por un conjunto de signos (letras, palabras, números, ondas, etc.), con lo cual las distintas elecciones de estos símbolos generarán mensajes diferentes. Esto confiere cierto grado de incertidumbre a la información contenida en el mensaje. Esta incertidumbre se describe por medio de la probabilidad de la elección de un mensaje, que depende de las probabilidades de elección de los símbolos que lo forman. A partir de esta probabilidad se puede obtener la cantidad de información contenida en el mensaje. Esta información puede representarse mediante bits. Luego, el número medio de bits necesarios para identificar cada símbolo del mensaje es $-\log_2 p$, donde p es la probabilidad de elección del símbolo. De acuerdo con esta expresión, cuanto más probable sea un mensaje, menos información proporcionará.

En el contexto de la clasificación con árboles de decisión, los símbolos son las clases y los mensajes son los subconjuntos de ejemplos de entrenamiento en los cuales se quiere dividir el conjunto de entrada de un determinado nodo. En este contexto la entropía se entiende como la cantidad media de información que se genera al escoger aleatoriamente un ejemplo del conjunto de datos del nodo a dividir, y después observar su clase.

Siendo p_j la probabilidad de que un ejemplo escogido aleatoriamente pertenezca a la clase j, la información que se obtiene al clasificar el ejemplo con dicha clase es $-\log_2 p_j$. Esta probabilidad se estima a partir de los datos de entrenamiento. Concretamente, p_j es la frecuencia relativa de la clase j en el conjunto de datos, es decir la proporción del número de ejemplos etiquetados con dicha clase.

La información media que se obtiene de la clasificación de cualquier ejemplo de un conjunto de datos de entrenamiento D es

$$H(D) = -\sum_{j=1}^{k} p_j \log_2 p_j \qquad (A.1)$$

donde k es el número de clases del problema. El valor obtenido es la entropía de clasificación del conjunto de datos D (H(D)).

Cuando todos los ejemplos del conjunto de datos pertenecen a una sola clase la entropía es 0 (para establecer este valor se define $0 \log_2 0 = 0$).

En cambio si las clases se reparten uniformemente entre los ejemplos se tiene que $p_1 = \cdots = p_j$ y la entropía vale 1, siendo este su valor máximo.

La cantidad de información obtenida de la división del conjunto total de ejemplos de entrenamiento en subconjuntos es

$$\sum_{i=1}^{m} p_i \, H(D_i) \qquad \text{(A.2)}$$

donde p_i es la probabilidad de que un ejemplo pertenezca al subconjunto D_i y m es el número de valores del atributo usado para la partición, y por tanto el número de subconjuntos a crear.

Al dividir el conjunto de ejemplos original en subconjuntos se crea una diferencia de entropía entre el conjunto original y los subconjuntos de la partición, esta diferencia se denomina ganancia de información:

$$H_{\text{gain}} = H(D) - \sum_{i=1}^{m} p_i \, H(D_i) \qquad \text{(A.3)}$$

La ganancia de información se utiliza típicamente para evaluar las posibles particiones del conjunto original. Pero es suficiente con utilizar la Ecuación A.2 para este fin, ya que la entropía de D es constante para todas las particiones evaluadas en el nodo actual.

B. Eye Tribe Tracker

B.1. Montaje

Antes de empezar a utilizar el Eye Tracker, este se debe montar en un trípode (proporcionado junto al dispositivo). Este soporte permite sostener la cámara de Eye Tracking a una altura correcta y además ajustar la posición de la misma respecto a los ojos de los usuarios, de modo que al finalizar el ajuste la cámara quede fija en dicha posición. Una vez, montada la cámara sobre el soporte, se debe conectar a un puerto USB 3.0.

Una vez conectado, se debe colocar la cámara montada en el trípode en una superficie horizontal lisa por debajo del monitor y centrado con respecto al mismo. En este proyecto se ha utilizado la misma mesa sobre la cual reposa el monitor. Según las indicaciones de The Eye Tribe la pantalla no debe tener un tamaño superior a las 24 pulgadas. De ser así, aseguran que el proceso de seguimiento ocular (Eye Tracking) no se ejecutará de forma óptima (es decir los datos pueden no ser correctos). No es aconsejable, por el mismo motivo, que la cámara se posicione por encima o a un lado del monitor. Además es importante que la cámara, junto con el soporte, estén colocados de tal manera que su posición y orientación no se vean alterados accidentalmente, ya que afectaría negativamente al proceso de captura.

B.2. Colocación usuario

En cuanto al usuario, debe colocarse correctamente frente al dispositivo Eye Tribe Tracker. De acuerdo con Eye Tribe, es recomendable colocar la cara de la persona entre 45 y 75 cm de distancia frente al monitor y lo más centrada posible con el mismo. Una vez que la persona está bien colocada se orientará la cámara, si es necesario, hacia la cara de la persona con la ayuda del trípode.

B.3. Área de seguimiento

El usuario debe estar colocado dentro de la "caja" de seguimiento del Eye Tracker. Esta "caja" es una región tridimensional dentro de la cual el dispositivo es capaz de realizar el seguimiento de los ojos del usuario. El tamaño de esta región varía según la velocidad de captura de datos del Eye Tracker: a mayor velocidad, más pequeña será dicha "caja".

En el caso del Eye Tribe Tracker, es capaz de capturar datos a velocidades de 30 y 60 Hz. Por tanto, la "caja" de seguimiento tendrá el tamaño máximo a 30 Hz y será menor a 60 Hz.

B.4. Software

B.4.1. EyeTribe Server

El EyeTribe Server es un servidor de aplicaciones. Se encarga de gestionar ciertos parámetros del Eye Tribe Tracker, así como la comunicación entre este y los clientes conectados al Eye Tracker. El cliente puede ser directamente el EyeTribe UI u otra aplicación que desee hacer uso de esta tecnología. Para que el cliente y el servidor puedan comunicarse, el EyeTribe Server establece una conexión TCP en localhost usando el puerto 6555, ya que el cliente debe ejecutarse en la máquina local a la cual está conectado el Eye Tracker (USB). Una vez conectados, el cliente y el servidor se envían información mutuamente de acuerdo con el protocolo de comunicaciones.

Este intercambio de información es transparente para las aplicaciones cliente que se comunican con el servidor, dado que dicha comunicación se oculta por medio de la API que proporciona el software de desarrollo de The Eye Tribe.

B.4.2. Protocolo de comunicaciones

El protocolo de comunicaciones consiste básicamente en el intercambio de mensajes en formato JSON. Los mensajes que intercambian cliente y servidor (tanto las solicitudes como las respuestas) se clasifican en tres categorías:

- Categoría tracker: Entre estos mensajes se incluyen todos los relacionados con la configuración del Eye Tracker, ya sea para obtener información acerca del dispositivo o de la pantalla (resolución de pantalla, estado de la calibración del Eye Tracker), como para establecer algunos parámetros (framerate a 30 o 60 fps). También se incluyen los mensajes utilizados para conocer el estado del Eye Tracker (dispositivo desconectado, dispositivo conectado y funcional, dispositivo conectado pero no funcional por incompatibilidad de puerto USB, etc). Por último, en esta categoría se incluyen los mensajes reservados para la obtención de los datos de la mirada del usuario. Nota: Como se ha indicado anteriormente es posible configurar en el servidor la velocidad de captura del dispositivo. Categoría calibración: En esta categoría se incluyen todos aquellos mensajes relacionados con el proceso de calibración. Estos mensajes tienen como objetivo establecer algunos parámetros del Eye Tracker para dicho proceso (establecer punto inicial, establecer punto final, eliminar última calibración del dispositivo), controlar su ejecución (iniciar o abortar el proceso) y obtener el resultado de la calibración cuando esta finaliza.

- Categoría pulso: La finalidad de estos mensajes es que cada cliente confirme al servidor que su conexión permanece activa. De esta forma el servidor puede detectar que clientes han perdido la conexión.

B.4.3. EyeTribe UI

El EyeTribe UI es una aplicación cliente que accede a las funciones del Eye Tracker a través del servidor (haciendo uso de la API). Consta de una interfaz gráfica sencilla que permite calibrar el dispositivo y ejecutar el proceso de Eye Tracking. Además, permite visualizar los datos obtenidos por el dispositivo "en vivo" en una consola e incluso almacenarlos en un archivo de texto.

Al iniciar la aplicación EyeTribe UI automáticamente se inicia también el EyeTribe Server, sin el cual no se puede enviar ni recibir información del Eye Tracker. Después se inicia o bien directamente la interfaz del EyeTribe UI o bien el modo demostración del Eye Tracker, en caso de que esta opción este activada (esto puede configurarse desde la interfaz gráfica del EyeTribe UI).

B.4.3.1. Interfaz principal

La interfaz principal del EyeTribe UI incluye en primer lugar una "caja de seguimiento". Se trata de una ventana dentro de la interfaz en la cual se incluye una representación gráfica de la posición de los ojos del usuario respecto del Eye Tracker. Es decir, se observará una animación que reflejará si un usuario se encuentra dentro del área de seguimiento de la cámara y en caso de estarlo, como de descentrado está o no respecto de la misma. Esta animación puede ser utilizada como guía, a la hora de cambiar la posición del usuario respecto del dispositivo si fuera necesario.

La animación puede presentar cuatro estados diferentes.

El primero de ellos corresponde a la imagen de dos ojos sobre un fondo verde. Esto quiere decir que el usuario está colocado de forma que sus dos ojos entran dentro del espacio de seguimiento del Eye Tracker.

El segundo estado que se puede presentar es el de un solo ojo sobre un fondo amarillo. Esta imagen indica que la cámara solo está captando uno de los ojos dentro del espacio de seguimiento.

En el siguiente estado aparecen dos ojos semi ocultos sobre un fondo rojo. Quiere decir que no se está detectando ninguno de los ojos del usuario.

El último estado directamente es un mensaje de texto indicando que se ha perdido la conexión con el dispositivo.

Justo debajo de esta ventana se encuentra el botón de calibración. Este botón inicia el proceso de calibración, el cual también se incluye en el modo demostración.

En la parte derecha de la ventana, se incluyen tres pestañas. Cada pestaña incluye varias opciones de configuración e información. La principal es la pestaña de calibración. En esta pestaña es posible cambiar algunos parámetros del proceso de calibración.

Por ejemplo, en la opción etiquetada como "Número de puntos" se puede especificar el número de puntos que se utilizarán en la pantalla de calibración. Esto es, el número de referencias que utilizará el Eye Tracker en pantalla para capturar las distintas posiciones de los ojos del usuario, que posteriormente utilizará para calcular las correspondientes coordenadas en el monitor.

En la opción "Selección del monitor" se puede seleccionar, de entre los mostrados en una lista desplegable, el monitor que se utilizará junto con el Eye Tracker. El monitor o monitores detectados automáticamente se mostrarán en dicha lista desplegable.

La opción "Tamaño de área" indica por defecto la resolución del monitor seleccionado anteriormente. No obstante es posible indicar una resolución menor, de modo que se pueda calibrar el dispositivo dentro de una sección reducida de la pantalla.

Las dos siguientes opciones "Alineación Horizontal" y "Alineación Vertical" se activan al seleccionar una resolución menor que la indicada por defecto en la anterior opción. La primera de estas dos opciones permite posicionar la región acotada en tres zonas a lo largo del eje horizontal, que son la parte izquierda, central o derecha de la pantalla. La segunda permite posicionar la región en tres zonas a lo largo del eje vertical, que son la parte superior, central o inferior.

De la pestaña de opciones lo más destacable es la posibilidad de abrir una consola en la que se imprimen los datos devueltos por el Eye Tribe Server. Dentro de esta consola existe la opción de pausar y reanudar la impresión de los datos, aunque no su recepción. También existe la opción de cambiar la salida de los datos a un archivo de texto, en lugar de mostrarlos en la consola.

B.4.3.2. Modo demostración

El modo demostración consiste en una serie de "pestañas" a pantalla completa entre las que se puede navegar utilizando los botones gráficos correspondientes a izquierda y derecha. En dicho modo se muestra una introducción al dispositivo de Eye Tracking y

un ejemplo de cómo se debe posicionar el usuario frente a la cámara y el monitor. Además se incluye una versión a pantalla completa de la "caja de seguimiento" de la interfaz principal del EyeTribe UI. Es decir la misma animación que representa la posición de los ojos del usuario respecto del Eye Tracker con los diferentes estados ya explicados. Las últimas "pestañas" están reservadas al proceso de calibración del dispositivo. Si se acepta la calibración, el modo demostración se cierra y se muestra la interfaz principal del EyeTribe UI.

B.5. Calibración

Antes de proceder con el seguimiento para un determinado usuario, se debe realizar un proceso de calibración del dispositivo Eye Tracker para dicho usuario. La razón principal es que cada persona tiene características oculares únicas, por tanto el Eye Tracker debe detectar y medir estas características con el objetivo de estimar la mirada de los usuarios con mayor precisión.

La calibración se puede realizar utilizando el software EyeTribe UI o a través de la API desde la aplicación cliente. En este proyecto se ha optado por la primera opción, ya que el método de calibración proporcionado por EyeTribe es bastante simple y suficientemente efectivo para el propósito de este proyecto.

El proceso de calibración consta de dos partes, la primera es la prueba de calibración y la segunda es una evaluación de la calibración.

Para realizar con éxito la calibración, el usuario debe seguir con la mirada un puntero circular de pequeño tamaño mientras este se desplaza y se detiene en un número determinado de puntos de la pantalla. En cada uno de los puntos en los cuales el puntero se detiene, el usuario debe mantener su mirada sobre él hasta que vuelva a cambiar de posición. Una vez que el puntero ha recorrido todas las posiciones al menos una vez, el proceso de calibración termina (es posible que el puntero requiera pasar más de una vez por algún punto si no se ha realizado bien la calibración sobre este). Este proceso dura aproximadamente 20 segundos en total y el puntero dedica aproximadamente 2 segundos a realizar las mediciones necesarias en cada una de las posiciones en la que permanece estático. Además el usuario debe evitar, en la medida de lo posible, mover la cabeza mientras dure el proceso para tener mayor probabilidad de obtener una calibración óptima.

Una vez completada la prueba de calibración, se muestra una pantalla de evaluación. En esta pantalla el usuario puede probar la calidad de la calibración realizada. Esto se realiza presentando en pantalla una serie de marcas circulares cuyas posiciones corresponden a aquellas utilizadas para la calibración. Si la calibración es buena, cuando el usuario centra su mirada o pasa muy cerca de dichas marcas, estas cambian de color. En la parte baja se muestra un medidor de calidad de la calibración, el cual indica dicha calidad utilizando un sencillo ranking. Cuanto más alto sea dicho ranking mayor precisión se puede esperar del seguimiento ocular.

En la Tabla 10 se especifican los seis posibles resultados del ranking y los mensajes del medidor.

Valoración	Mensaje	Descripción
5 estrellas	Perfecto	Resultado de calibración óptimo. No es necesario recalibrar. (< 0.5^0 de error en ángulo visual).
4 estrellas	Bueno	La calibración es bastante buena para el seguimiento. (< 0.7^0 de error en ángulo visual).
3 estrellas	Moderado	La calibración es aceptable. No obstante, es aconsejable volver a calibrar la cámara. (< 1^0 de error en ángulo visual).
2 estrellas	Pobre	Este resultado no es suficientemente bueno para realizar un seguimiento ocular preciso. Es muy aconsejable recalibrar. (< 1.5^0 de error en ángulo visual).
1 estrella	Recalibrar	El resultado no es bueno. Se debe recalibrar el dispositivo.
Ninguna estrella	Sin calibrar	El dispositivo no se ha calibrado.

Tabla 10. Ranking y mensajes de la calidad de la calibración

Si el Eye Tracker está calibrado como mínimo a un nivel moderado, las coordenadas en pantalla de la mirada del usuario se calculan con un error medio que varía entre 0.5^0 y 1^0 en el ángulo visual. Suponiendo que el usuario este sentado a 60 cm de la pantalla, el error medio correspondiente en pantalla varía entre 0.5 y 1 cm.

Justo debajo del medidor se incluyen también dos botones: Recalibrar y Aceptar. El botón de Recalibrar vuelve a iniciar el proceso de calibración antes explicado en caso de que el usuario no esté contento con el resultado. El botón de aceptar cierra el proceso guardando la calibración realizada y abre la interfaz gráfica del EyeTribe UI.

Se debe tener en cuenta una cuestión muy importante. Una vez realizada y aceptada la calibración se debe tener mucho cuidado de no mover el Eye Tracker de su posición. Si ocurriese esta situación, las mediciones y estimaciones del Eye Tracker quedarían alteradas negativamente y sería necesario recalibrar el dispositivo.

B.6. Datos oculares

El Eye Tracker envía en tiempo real las capturas realizadas durante el proceso de seguimiento ocular del usuario, utilizando la API de comunicación y a una frecuencia regulada por el servidor (en este caso 30 fps). Cada captura contiene principalmente los datos estimados de la mirada del usuario, junto con otros datos relacionados con el estado y la calidad del seguimiento ocular.

A continuación, se describen algunos de los datos más relevantes utilizados en el proyecto.

B.6.1. Estado de la captura

Es un número natural que indica el estado del proceso de captura. Concretamente indica si el usuario se encuentra dentro del espacio de seguimiento de la cámara y la calidad de dicho seguimiento.

B.6.2. Coordenadas de la mirada (un ojo)

Estas coordenadas se refieren al punto dentro de la pantalla que el usuario está mirando actualmente. Estos puntos se definen usando los ejes x e y del sistema de coordenadas de dos dimensiones, cuyo origen se encuentra en el punto más alto y a la izquierda de la pantalla. Ambas coordenadas, x e y, se miden en píxeles en la pantalla. De modo que, para estimar el punto sobre la pantalla que el usuario está mirando en un momento determinado, se calculan las coordenadas x e y en píxeles del mismo.

Dado que el origen de coordenadas se encuentra en la parte más alta y a la izquierda de la pantalla, el eje y está orientado de forma que el origen queda en el extremo superior de la misma y el valor más alto del eje en el extremo inferior (número máximo de píxeles de alto). Por tanto, los valores del eje y son más positivos en sentido descendente, con lo cual el eje está orientado de manera inversa a la forma estándar. En cambio, el eje x si está orientado de la forma más común, es decir los valores son más positivos de izquierda a derecha.

Cabe mencionar que estas coordenadas se calculan para cada ojo de manera individual. Es decir, el Eye Tracker devolverá las coordenadas x e y para el ojo izquierdo y para el ojo derecho.

B.6.3. Coordenadas de la pupila (un ojo)

Estas coordenadas corresponden a la posición relativa de las pupilas de una persona respecto del sensor del Eye Tracker. Esta posición se define mediante los ejes x e y de un sistema de coordenadas de dos dimensiones. Estos ejes están orientados de igual forma que los utilizados para las mediciones en pantalla, como se ha explicado anteriormente. Las coordenadas x e y se expresan usando valores numéricos normalizados y relativos al sensor del Eye Tracker.

Estas coordenadas, al igual que las de la mirada en pantalla, también se calculan para cada ojo de manera individual.

B.6.4. Diámetro de la pupila (un ojo)

El Eye Tracker calcula el diámetro de pupila de los ojos. El diámetro se expresa en milímetros y se calcula de manera individual para cada ojo.

B.6.5. Coordenadas de la mirada (ambos ojos)

Estos datos no se calculan de igual forma que la expuesta en el apartado correspondiente a las coordenadas de cada ojo. En este caso dado que ya están calculadas estas coordenadas, el Eye Tracker devuelve el resultado de hacer la media entre los dos ojos. Es decir, por un lado se realiza la media entre la coordenada x del ojo izquierdo y la coordenada x del ojo derecho, y por otro lado la media de la coordenada y del ojo izquierdo y la coordenada y del ojo derecho.