Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Detección de comportamientos erráticos o anómalos en el uso de dispositivos para la autogestión de ataques psicóticos

Detection of erratic or abnormal behaviors in the use of devices for the psychotic attacks self-management

Victoria Quintana Martí

La Laguna, 6 de julio de 2020

D. **Iván Castilla Rodríguez** con N.I.F. 78.565.451-G profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Rafael Arnay del Arco**, con N.I.F. 78.569.591-G profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Detección de comportamientos erráticos o anómalos en el uso de dispositivos para la autogestión de ataques psicóticos"

ha sido realizada bajo su dirección por D. **Victoria Quintana Martí**,

con N.I.F. 43.382.763-W.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2020.

Agradecimientos

A mis tutores Iván y Rafael, por ayudarme y aconsejarme en todo momento.

A mis amigos y familiares, por apoyarme y estar siempre conmigo en los momentos más difíciles.

Licencia

Resumen

Cuando un individuo sufre un ataque psicótico pierde el contacto con la realidad, y debido a esto, se le hace realmente difícil detectar si lo que está pasando es real o está sufriendo un ataque. En ese momento, la persona puede realizar acciones peligrosas, ya que está percibiendo una realidad alterada y puede actuar en consecuencia a lo que esté sintiendo. Uno de los escenarios donde puede ponerse de manifiesto este tipo de conductas, es en la interacción con el ordenador o el móvil, ya que actualmente la tecnología está muy presente en la vida cotidiana, y si se dispusiera de una herramienta que fuese capaz de detectar esta situación de riesgo, se podría avisar al individuo o a terceras personas.

De este modo, surge el desarrollo de la aplicación que se va exponer en este Trabajo de Fin de Grado, cuyo objetivo es detectar el comportamiento anómalo en el uso del ordenador mientras se está teniendo el ataque.

La aplicación distingue un comportamiento normal y uno anómalo mediante el análisis del comportamiento del usuario con el teclado y el ratón al usar el ordenador. Con esta información, la aplicación es capaz de distinguir cuándo se sale de los valores normales establecidos, y avisar al usuario mediante una notificación. Además, con la información recogida sobre el uso del teclado, se realiza un análisis semántico de las palabras escritas y se clasifican usando Machine Learning, para así, poder detectar el tipo de emociones que está sintiendo el usuario al escribir las frases y tener más información para la detección de un comportamiento anómalo. Por otro lado, con los datos recogidos del ratón, se realiza un mapa de calor para ver por dónde se mueve el usuario a través de la pantalla. Toda la información sobre la actividad del usuario que haya recopilado la aplicación, podrá ser consultada en cualquier momento en un cuadro de mandos por el propio usuario.

Palabras clave: C#, Python, Machine Learning, anómalo, ratón, teclado, psicótico, lenguaje natural, análisis semántico, modelo, comportamiento, emociones, interacción.

Abstract

When an individual suffers a psychotic attack, he loses contact with reality, and due to this, it becomes really difficult to detect if what is happening is real or is having an attack. At that time, the person can perform dangerous actions, since he is perceiving an altered reality and can act accordingly to what he is feeling. One of the scenarios where this type of behavior can be manifested is in the interaction with the computer or mobile phone, since technology is currently very present in everyday life, and if it had a tool that is capable of detecting this risk situation could warn the individual or third parties.

In this way, the development of the application that is going to be exposed in this Final Degree Project arises, whose objective is to detect abnormal behavior in the use of the computer while it is having the attack.

The application distinguishes between normal and abnormal behavior by analyzing user behavior with the keyboard and mouse when using the computer. With this information, the application is able to distinguish when gets out from the established normal values, and notify the user by means of a notification. In addition, with the information collected on the use of the keyboard, a semantic analysis of the words written and classified using Machine Learning is carried out, in order to detect the type of emotions the user is feeling when writing sentences and have more information to stopping abnormal behavior. On the other hand, with the data collected from the mouse, a heatmap is made to see where the user moves through the screen. All the information of the activity of the user that the application has collected, may be consulted at any time in a control panel by the user himself.

Keywords: C#, Python, Machine Learning, abnormal, mouse, keyboard, psychotic, natural language, semantic analysis, model, behaviour, emotions, interaction.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1. Estado del Arte

1.1.1. Marco

La psicosis [0] es una enfermedad mental grave que afecta algunas funciones cerebrales tales como el pensamiento, la percepción, las emociones y la conducta. Afecta aproximadamente a un 1% de las personas durante su vida independientemente del sexo, raza y clase social. Uno de los síntomas más comunes es el de perder el contacto con la realidad, y este síntoma puede ser muy peligroso, ya que las personas que lo padecen perciben, durante un periodo de tiempo, una realidad alterada que no es la que el resto de personas viven pero que ellos mismos la creen cierta, y esto puede causarles angustia y nerviosismo. Por este motivo, con el fin de evitar que ocurra algo grave, se pensó en la creación de una aplicación que pudiera detectar cuándo se está teniendo un ataque, mediante la monitorización del teclado y del ratón al usar el ordenador, y a partir de ellos, analizar el comportamiento del usuario.

1.1.2. Motivos, Razones y Problemas

Actualmente no existe una aplicación como tal que analice los comportamientos de una persona que tiene ataques psicóticos y que los clasifique como anómalos o no, pero sí se han encontrado investigaciones como en el informe de Detecting Abnormal User Behavior Through Pattern-mining Input Device Analytics [1], en el que se estudia cómo detectar los patrones de uso del ordenador y se analiza por qué se tiene dicho comportamiento. También existen aplicaciones que controlan el comportamiento del usuario en el PC pero para vigilar al usuario como por ejemplo en empresas, y ver cuál es el comportamiento de los empleados. La mayoría de aplicaciones de este estilo suelen ser de pago como FireWorld [2] o AeroAdmin [3]. Aparte de aplicaciones ya realizadas, también se encontraron pequeños programas en github en los que se monitoriza el movimiento del ratón o del teclado del usuario, como el de jnativehook [4], pero no se encontraron aplicaciones para el uso que en este TFG se le quiere dar.

Por otro lado, existen aplicaciones para ver cómo está de ánimos un paciente que tiene este tipo de enfermedades como Sardaa [5], pero en estas aplicaciones no se puede saber si el paciente está sufriendo un ataque. Por ello, con la aplicación que se está proponiendo, se va un paso más allá y se podría evitar situaciones de peligro.

1.2. Objetivos y fases

1.2.1. Objetivos

El objetivo principal del proyecto es el desarrollo de una aplicación que sea capaz de detectar cuándo se está sufriendo un ataque psicótico a través del uso del ordenador.

Para poder analizar el comportamiento de un usuario con un ordenador, se debe obtener, de alguna forma, cómo se está comunicando con él para así poder distinguir cuándo está haciendo un uso anormal y cuándo no. La forma más común de comunicarse y utilizar un ordenador es por medio del teclado y el ratón. Por ello, uno de los principales objetivos, es encontrar la forma de obtener el uso del ratón y el teclado por parte del usuario.

Una vez se obtenga cómo el usuario usa el teclado y el ratón en el ordenador durante un periodo de tiempo, se podría hacer un perfil de cómo se utiliza en un estado de normalidad y en este caso, cuándo se salga de este estado, se consideraría que está usando de forma anormal. Por ello, el segundo objetivo, es que una vez recogido los parámetros de teclado y ratón, se puedan detectar patrones en el uso de ambos periféricos y poder hacer una clasificación entre un uso normal y anormal.

Las entradas del teclado recogidas pueden emplearse para un nuevo objetivo, que consistirá en analizar semánticamente las frases que escribe el usuario y clasificarlas dependiendo de sus emociones, para así, detectar si lo que está escribiendo está dentro de la normalidad o no.

Finalmente, el último objetivo será realizar un mapa de calor con la entrada del ratón, para así, poder analizar en qué sitios de la pantalla el usuario pasa la mayor parte del tiempo.

Una vez se obtengan todos estos parámetros, se desarrollará un cuadro de mandos en el que el usuario pueda analizar en qué momento se está haciendo un uso anormal del ordenador.

1.2.2. Fases

El desarrollo del proyecto se ha llevado a cabo en siete fases:

1. **Documentación:** Durante unas semanas se realizaron varias investigaciones buscando trabajos similares al que se iba a realizar y formas de capturar el teclado y el ratón, para así, tener una base por donde empezar.
2. **Detección del teclado y el ratón:** En esta fase, el objetivo era obtener la entrada del teclado y del ratón de forma que se pudiera parametrizar y trabajar con estos datos.

3. **Detección de patrones anómalos del teclado y del ratón:** Para esta fase, se creó un perfil del usuario usando la aplicación como lo haría de forma “normal”, cuando no está sufriendo un ataque, para así poder fijar los valores que se consideran “normales”.
4. **Clasificación de patrones anómalos:** Una vez obtenidos los valores que se consideran “normales”, se fijaron los límites para detectar que cuando se sale de ellos está fuera de lo “normal” y avisar al usuario.
5. **Clasificación de frases por emociones:** La siguiente fase consistió en la realización de un modelo que clasifica las frases que el usuario escribe por teclado por la emoción que evocan.
6. **Realización de cuadro de mandos y mapa de calor:** Después de detectar y clasificar el uso del ratón y el teclado, se elaboró, dentro del programa creado, una ventana en la que se pueden ver todos los parámetros que se están analizando y en qué momento está saltando alguna alarma. En esta ventana, se incluyó un mapa de calor para complementar y tener más información acerca de la actividad del usuario.
7. **Testeo:** en esta fase se comprobó que la aplicación funcionaba correctamente.

Capítulo 2. Tecnologías utilizadas

Durante el desarrollo de la aplicación que se expone en este Trabajo de Fin de Grado, se han utilizado varias tecnologías, las cuales se detallarán a lo largo de este capítulo.

2.1. C#

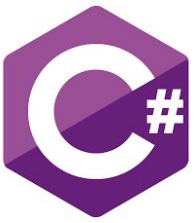

Figura 2.1 Logo de C#

C# [6] es un lenguaje de programación multiparadigma que fue desarrollado y estandarizado por Microsoft y que, al cabo de un tiempo, fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). Su sintaxis básica

deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

Es un lenguaje orientado a objetos, que fue diseñado con el objetivo de permitir a los desarrolladores crear una multitud de aplicaciones que fueran ejecutadas en .NET Framework [7], que se trata de una tecnología que admite la compilación y ejecución de aplicaciones y servicios web XML. Por ello, C# se caracteriza por una sintaxis sencilla, con seguridad de tipos y orientado a objetivos.

Para desarrollar la mayor parte de la aplicación, se escogió el lenguaje de programación C# debido a que para desarrollar aplicaciones de escritorio de Windows, este lenguaje es muy seguro y sólido. Además, el desarrollo de aplicaciones con este lenguaje puede resultar más sencillo ya que era uno de los objetivos cuando se creó para su uso en .NET.

2.1.1. Librerías externas

En el desarrollo de la aplicación usando C# se utilizaron varias librerías externas que fueron cruciales para poder completar los objetivos que se propusieron.

2.1.1.1. globalmousekeyhook

Figura 2.2 Logo de globalmousekeyhook

La aplicación desarrollada está la mayor parte del tiempo en background, y debido a esto, se necesitaba de alguna forma obtener la actividad del teclado y el ratón sin tener la aplicación abierta, si no que se recogiera la información teniendo la aplicación en segundo plano. Con las librerías propias de C#, esto no era posible ya que se tenía que tener la aplicación en primer plano. Por ello, se utilizó la librería globalmousekeyhook [8], que permite justo lo que se ha mencionado anteriormente, obtener en todo momento, con la aplicación en segundo plano, la actividad del teclado y del ratón.

La librería rastrea los clics y movimientos del ratón y las pulsaciones del teclado para generar eventos .NET comunes con **`KeyEventArgs`** y **`MouseEventArgs`**. Esto permite que después de haberse lanzado el evento, se pueda recuperar fácilmente cualquier información que se necesite. Entre la actividad que se puede recoger, las que se han utilizado para la aplicación son:

- Coordenadas del ratón

- Botones del ratón que se han presionado
- Desplazamiento de rueda del ratón
- Teclas del teclado que se han pulsado

2.1.1.2. Newtonsoft

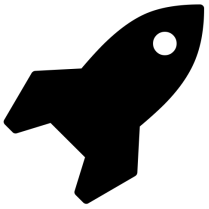

Figura 2.3 Logo de NewtonsofIt

Para recoger toda la actividad con el uso del teclado y ratón, se utilizó JSON [9], que es un formato de texto sencillo para el intercambio de datos. De esta forma, se creó una estructura ordenada, la cual contiene toda la información recogida en el periodo de tiempo que el usuario haya tenido la aplicación activa.

En C#, para poder utilizar y crear un JSON, se utiliza la librería Newtonsoft [10], la cual te permite crear a partir de texto un fichero .JSON de forma sencilla. Para crear a partir de un texto, una estructura JSON se utiliza la Serialización que lo que hace es que convierte un objeto .Net personalizado en una cadena JSON. Para invocar este método se utiliza la clase **`SerializeObject`**, a la cual se le pasa la cadena a serializar y si se va a poner con formato indentado.

2.2. Python

Figura 2.4 Logo de Python

Python [11] es un lenguaje de programación interpretado y multiplataforma que apuesta por la simplicidad, versatilidad y rapidez de desarrollo. Soporta la

programación imperativa y la orientación a objetos por lo que se trata de un lenguaje multiparadigma.

Se escogió este lenguaje en la versión 3.7 para desarrollar el script de clasificación del texto por emociones. En un primer momento, se intentó clasificar los textos con una librería de Machine Learning de C# pero no funcionaba como se esperaba, y por ello, se cambió a Python ya que tiene una librería llamada Scikit-learn para la clasificación y análisis de datos. En el capítulo 2.1.1.2 se comentará más en detalle sus características.

Por otro lado, también se escogió este lenguaje para la creación de los gráficos que se muestran en el Cuadro de Mandos con la actividad del usuario, debido a que permitía realizar estos gráficos de forma rápida y sencilla.

2.2.1. Librerías externas

En este apartado, se detallarán las librerías utilizadas en el desarrollo de los scripts en Python.

2.2.1.1. Matplotlib

Figura 2.5 Logo de matplotlib

Matplotlib [12] es una librería para crear visualizaciones estáticas, animadas e interactivas en Python. Mientras que en otros lenguajes el desarrollo de gráficas o visualizaciones interactivas se hacía pesado y difícil de comprender, con matplotlib se hacía mucho más sencillo y rápido.

En este proyecto se ha utilizado para crear las gráficas del Cuadro de Mandos con la actividad del usuario. Por otro lado, también se utilizó para crear el mapa de calor a partir de las coordenadas del ratón del usuario.

Figura 2.6 Gráficos radiales

2.2.1.2. Scikit-learn

Figura 2.7 Logo de Scikit-learn

Scikit-learn [13] es una librería para el aprendizaje automático que incluye varios algoritmos de clasificación y regresión. Además, ofrece métodos para separar el conjunto original de datos en dos subconjuntos, uno de entrenamiento y otro de prueba, para así poder obtener la proporción de cada uno con respecto al original.

Para este proyecto se ha utilizado esta librería, ya que en la clasificación del texto que introduce el usuario por teclado por las emociones que evoca, se hacía imprescindible encontrar una librería que permitiera crear un modelo de clasificación fiable y seguro.

2.2.1.3. NLTK

Figura 2.8 Logo de Natural Language Toolkit

Natural Language Toolkit [14] es una plataforma de procesamiento natural del lenguaje para Python que proporciona demostraciones gráficas y datos de muestra junto con recursos léxicos como Wordnet, que contiene una base de datos extensa para el léxico en inglés, así como un conjunto de bibliotecas de procesamiento multilingüe.

Este módulo cuenta con una serie de corpus y colecciones de textos en diferentes idiomas que facilitan el uso y agilizan en gran medida las pruebas. También, posee múltiples métodos de cálculo de frecuencias y probabilidades de búsquedas de términos y de textos, así como varias características de procesamiento. Entre ellas encontramos la tokenización de palabras, la segmentación del texto en frases, el análisis morfológico, la lematización y el stemming.

En la aplicación desarrollada, se ha hecho uso de esta librería para la lematización y el stemming del dataset que se utilizó en el modelo de clasificación. En el capítulo 3.4.2.1.1. se explicará más en detalle el desarrollo utilizando esta librería.

2.2.1.4. Googletrans

Figura 2.9 Logo de Googletrans

Googletrans [15] es una librería implementada por Google Translate API para python que usa la Google Translate Ajax API para hacer llamada métodos para poder hacer traducciones. Es muy rápida ya que usa los servidores de translate.google.com y es capaz de detectar automáticamente el idioma escrito. Por otro lado, hay que tener cuidado con cuántas peticiones se hacen al servidor, ya que si se llegan a hacer demasiadas puede dejar de funcionar por haber superado las peticiones permitidas.

Esta librería se ha utilizado en el proyecto para traducir las frases que escribe el usuario a inglés. Las frases se traducen al inglés ya que es el idioma del dataset que se ha utilizado para entrenar y testear el modelo que se utiliza en la clasificación de las frases por emociones.

2.2.1.5. Pandas

Figura 2.10 Logo de pandas

Pandas [16] es una librería para la manipulación y análisis de datos para Python. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. Permite crear tipos de datos DataFrame (marco de datos) para la manipulación de datos con indexación integrada.

Se ha utilizado esta librería para la creación de DataFrames para los gráficos radiales.

2.2.1.6. Pickle

Figura 2.11 Logo de pickle

Pickle [17] es una librería que permite guardar y cargar los modelos que se entrenan para después usarlos en la clasificación de datos. Esto lo que permite es que no se tenga que estar entrenando el modelo varias veces para poder hacer la clasificación, si no que basta con tener el modelo guardado y luego abrirlo para realizar la clasificación.

En este proyecto se ha utilizado para guardar el modelo de clasificación de textos por emociones para así, utilizarlo en otro script de forma cómoda sin tener que volverlo a entrenar de nuevo.

Capítulo 3. Desarrollo de la aplicación

3.1. Estructura de la aplicación

La aplicación que se ha desarrollado para este Trabajo de Fin de Grado, consta de un programa principal que recoge la actividad del usuario con el uso del teclado y el ratón, un script que analiza el texto escrito con el teclado para luego clasificarlo por emociones, otro script que se encarga de realizar gráficos con la información recogida en el programa principal y un último script que crea un mapa de calor con las coordenadas del ratón.

El programa principal (**MKHOOK**) es el que ejecuta la aplicación en su totalidad. Este programa está escrito en C# y se encarga de recolectar toda la actividad del usuario con el teclado y el ratón. Está ejecutándose siempre en segundo plano recogiendo cada cierto tiempo la actividad del usuario y escribiéndola en un fichero .JSON. La aplicación, a la vez que ejecuta el programa principal, lanza los scripts escritos en Python mediante hilos en paralelo. Estos scripts son el analizador de emociones y la creación de la gráfica correspondiente (**`graphics_emotions.py`**), la creación de los gráficos con la información del teclado y el ratón (**`graphics_mk.py`**) y la creación del gráfico con el mapa de calor (**`heatmap.py`**). La comunicación con la aplicación principal es mediante ficheros, y través de ellos, los scripts recogen la información de la actividad del usuario para luego, poder crear los gráficos correspondientes y mostrarlos en el programa principal.

En la figura que se muestra a continuación, se puede observar el diagrama del funcionamiento de la aplicación.

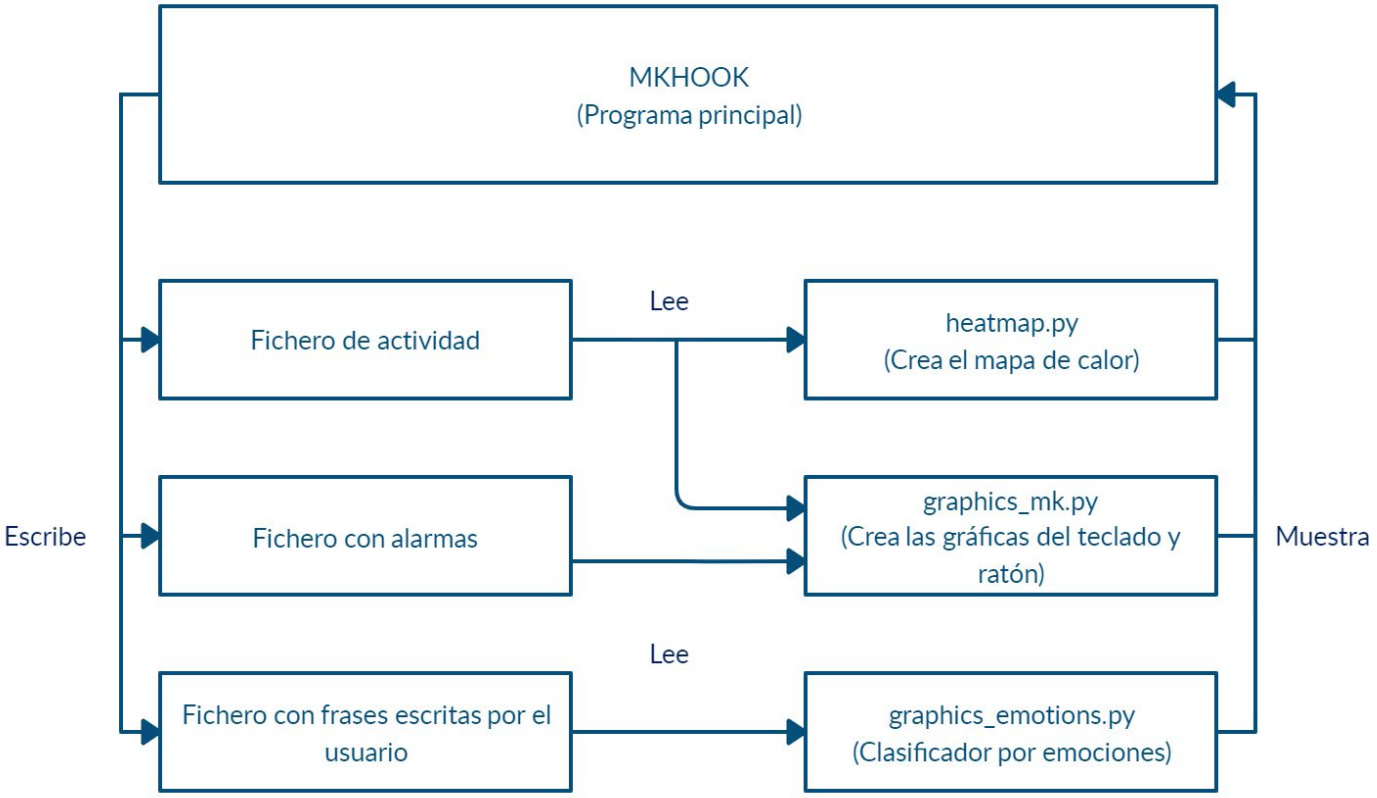

Figura 3.1 Diagrama con funcionamiento de la aplicación

3.2. Programa principal MKHOOK

3.2.1. Organización del código

La aplicación principal posee un código con una estructura simple y orientada a objetos. Se divide en varias clases y cada una tiene una funcionalidad diferente dentro del programa.

3.2.1.1. Clase principal

3.2.1.1.1. App.cs

`App.cs` es la clase clase principal que orquesta todas las funciones que realiza el programa. Cuando se ejecuta, aparece un icono en la bandeja de notificación, igual que el que se muestra en la siguiente figura, que indica que el programa se está ejecutando en background.

Figura 3.2 Icono de la aplicación

Si se pulsa con el click derecho encima del icono, se podrán escoger entre las siguientes opciones:

- **Config:** al pulsar en esta opción se desplegará un formulario de la propia clase, que permitirá cambiar cada cuánto tiempo quiere que se realice el muestreo de ratón y de teclado en general.
- **Alarm:** si se escoge esta opción, se desplegará un formulario de la clase **`AlarmForm.cs`** que contiene un resumen con los parámetros que se están midiendo en cada momento. También se puede establecer el valor de la alarma para indicar qué valores se consideran anómalos y que salte un aviso si se han traspasado. En este formulario también aparecen los gráficos con la actividad del usuario.
- **Exit:** pulsando en esta opción dejaremos de ejecutar la aplicación.

Una vez el programa se esté ejecutando, se creará un objeto de la clase **`Events.cs`** que estará recogiendo todos los parámetros necesarios para poder detectar si se está produciendo alguna anomalía en el comportamiento del usuario. En el momento en el que la aplicación comienza a ejecutarse, se lanzarán los scripts que realizan la clasificación del texto por emociones, la creación de los gráficos radiales y la creación del mapa de calor en varios hilos paralelos al de la propia ejecución.

Aparte de ser la clase principal, esta clase actúa de formulario en el que se indica cada cuánto quiere que se recoja la actividad del teclado y del ratón y se escriba en el fichero de actividad, y cada cuánto quiere que se rastreen las coordenadas del ratón, tal y como se muestra en la siguiente imagen.

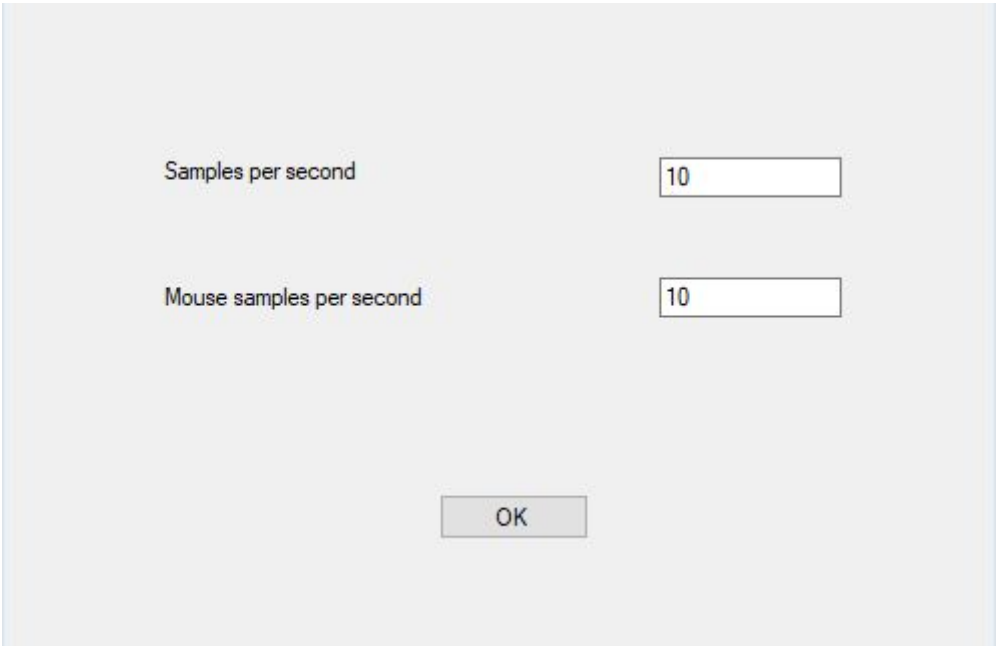

Figura 3.3 Ventana para actualizar el tiempo de escritura en el fichero y el tiempo de rastreo de las coordenadas del ratón

3.2.1.2. Monitorización de la pantalla, captura del teclado y ratón

Para la monitorización de la pantalla del usuario, se utiliza la clase **`Events.cs`** que es la encargada de recolectar toda la actividad que se realiza con el teclado y el ratón. Por otro lado, los parámetros que van a medirse en la captura de dichos periféricos se definirán en las clases **`Mouse.cs`** y **`Keyboard.cs`**.

3.2.1.2.1. Event.cs

`Events.cs` es la clase donde se recoge la actividad del teclado y del ratón mediante eventos, para luego, escribir esa información en un fichero .JSON.

Los eventos son sucesos que ocurren en un programa, definido por el usuario o provocado por él mismo. En este caso, estos eventos estarán controlados por la librería globalmousekeyhook [8]. Para ello, lo primero que hay que hacer es crear un objeto **`IKeyboardMouseEvents`**, que es el que gestionará los eventos y luego crear las suscripciones. Una suscripción a un evento permite escribir código personalizado que llama al evento en cuestión. Por ejemplo, se puede realizar una suscripción a un evento **`KeyDown`** que lo que haga es que cuando se pulse una tecla se recoja qué tecla ha sido pulsada.

En este caso, se creó una función **`Subscribe`** que se suscribe a todos los eventos de teclado y ratón. Estos eventos son:

- **`OnKeyDown:`** recoge el número de teclas que va pulsando el usuario y cuáles son para luego escribirlas en un fichero.
- **`OnKeyUp:`** indica cuándo el usuario ha dejado de pulsar una tecla.

- **`OnMouseUp:`** en este evento se recoge cuándo el usuario ha dejado de hacer click con el ratón.
- **`OnMouseClick:`** recoge el número de veces que un usuario hace click con el ratón.
- **`OnMouseDoubleClick:`** indica el número de veces que un usuario hace doble click con el ratón.
- **`HookManager_MouseMove:`** recoge las coordenadas en las que se encuentra el ratón. Con este dato, luego se calculan los desplazamientos que realiza el usuario por la pantalla.
- **`HookManager_MouseWheelExt:`** en este evento se indican las veces que el usuario hace scroll con el ratón para, posteriormente, medir el cambio de dirección del scroll en un tiempo determinado.

```
private void Subscribe(IKeyboardMouseEvents events)
{
    m_Events = events;
    m_Events.KeyDown += OnKeyDown;
    m_Events.KeyUp += OnKeyUp;

    m_Events.MouseUp += OnMouseUp;
    m_Events.MouseClick += OnMouseClick;
    m_Events.MouseDoubleClick += OnMouseDoubleClick;
    m_Events.MouseMove += HookManager_MouseMove;
    m_Events.MouseWheelExt += HookManager_MouseWheelExt;

}
```

Figura 3.4 Código del método Susbscribe de la clase Events.cs

Una vez se hayan terminado de manejar los eventos, es importante cancelar los eventos a los que estaba suscritos. Para ello, se hará de la misma forma que la suscripción pero cambiando el "+" por el "-", como por ejemplo, con el evento **`OnKeyDown`** que se realizaría de la forma que se indica en la siguiente figura.

```
m_Events.KeyDown -= OnKeyDown
```

Figura 3.4 Línea de código para desuscribirse de un evento

Aparte de los eventos de teclado y ratón, en esta clase existe otro evento **`ElapsedEventHandler`** de la clase `Timer`, que permite tener un temporizador que controla cada cuánto tiempo se escribe la actividad del usuario en el fichero de actividad .JSON y cada cuánto tiempo se guardan en un objeto de la clase **`Mouse.cs`** las coordenadas del ratón. Todas las coordenadas del ratón no se

guardan cada vez que se dispara el evento, ya que serían demasiados datos. Por ello, para limitar esta información, cada segundo se localiza dónde se encuentra el ratón y se guardan dichas coordenadas, para luego poder calcular la distancia euclídea y determinar los movimientos del usuario.

3.2.1.2.2. Mouse.cs

En la clase **`Mouse.cs`** es donde se establecen los parámetros que se van a medir del ratón, para así, poder determinar si el usuario está teniendo un comportamiento anómalo. En la clase **`Events.cs`** se crea un objeto de esta clase para que cuando se dispare un evento de tipo ratón se recoja la actividad y se guarde en un objeto de esta clase. Dependiendo de los atributos que tiene la clase como el número de clicks, el scroll que realiza con el ratón y las coordenadas de posición, se definieron una serie de medidas/métricas a tener en cuenta para analizar el comportamiento del usuario:

Acciones/Interacciones		**Medidas/Métricas**
Ratón	Nº de pulsaciones	Pulsación de clicks del ratón en un tiempo determinado.
	Scroll	Medir el cambio de dirección del scroll en un tiempo determinado.
	Desplazamientos del ratón	Cálculo de distancia euclídea en un tiempo determinado.

Tabla 3.1 Medidas del ratón

Todas estas medidas se calculan a través de varios métodos de la propia clase, para así sacar un valor que luego pueda ser comparado y determinar si se está teniendo un comportamiento fuera de lo normal. La clasificación en “anómalo” o “no anómalo” se realiza en la clase **`AlarmForm.cs`** la cual se explicará detalladamente en el capítulo 3.2.1.4.1.

3.2.1.2.3. Keyboard.cs

Al igual que la clase **`Mouse.cs`**, la clase **`Keyboard.cs`** establece los parámetros que se van a medir del teclado para poder determinar el comportamiento del usuario. En **`Events.cs`** se creará un objeto de esta clase, para así, cada vez que ocurra un evento de tipo Teclado se guarde la actividad realizada en este objeto.

La clase cuenta con una serie de atributos que le permite guardar toda la información necesaria para luego clasificar el comportamiento del usuario:

- **`pressedKeys`**: Número de teclas pulsadas.

- **`backSpaceKey`**: Número de veces que se ha pulsado la tecla ESC.
- **`twoKeyPressed`**: Número de veces que se ha pulsado dos teclas a la vez.
- **`words`**: Teclas que se han pulsado.
- **`keydown`**: Booleano para controlar si se han pulsado dos teclas a la vez. Cuando ha pasado un tiempo muy muy corto entre que se pulsó una tecla y otra se considerará que se han pulsado dos a la vez.
- **`startTime`**: Tiempo en el que se inicia la medición.
- **`timeElapsed`**: Tiempo que ha transcurrido.

Con estos atributos, se realizaron varios métodos que permitían calcular las medidas/métricas que se detallan a continuación:

Acciones/Interacciones		Medidas/Métricas
Teclear	Velocidad de escritura	Pulsación de teclas en un tiempo determinado.
	Veces que se pulsa la tecla retroceder	Nº de veces que se pulsa retroceder en un tiempo determinado.
	Pulsación simultánea de teclas	Nº de veces que se pulsan dos teclas simultáneamente en un tiempo determinado.

Tabla 3.2 Medidas del teclado

Los valores obtenidos de las medidas/métricas que aparecen en la tabla se utilizarán posteriormente en la clase **`AlarmForm.cs`** para así poder clasificar el comportamiento del usuario.

3.2.1.3. Informes de actividad

Toda la actividad que se recoge del teclado y del ratón, cada cierto tiempo, se escribe en dos ficheros. Un fichero .JSON con toda la actividad del teclado y el ratón, y un fichero en el que solo se guarda las teclas que ha pulsado el usuario.

3.2.1.3.1. JsonClass.cs

En la clase **`JsonClass.cs,`** se define la estructura que tendrá el fichero JSON que contiene toda la información acerca de los parámetros que se van a medir, para así, poder detectar si se está produciendo una anomalía o no en el comportamiento del usuario.

En la estructura del .JSON lo primero que se definen son el periodo de tiempo en el que se recoge la actividad del usuario y el periodo en el que se recogen las

coordenadas del ratón. Esta información, se indica al principio del fichero con los objetos "**TimeActivityPerSecond"** y "**MouseSamplePerSecond"**. Si en algún momento se cambia alguno de estos datos en el formulario que se indicaba en la Figura 3.3, se escribirá un nuevo fichero y se comenzará a recoger la actividad del usuario dependiendo del tiempo que se haya indicado.

A continuación, se define una lista o array con la actividad del usuario cuya clave es **"Activity".** Dentro de esta lista, lo primero que se define es la fecha y hora en la que se ha realizado la captura de la actividad con la clave **"Time".**

Después, se define la actividad del usuario con el teclado con la clave **"Keyboard"**. Aquí se recogen las veces que ha pulsado una tecla el usuario en la clave **"PressedKeys"**, las veces que se ha pulsado la tecla de retroceder en **"BackSpaceKey"** y las veces que se han pulsado dos teclas a la vez en **"TwoPressedKeys".**

Justo después del objeto **"Keyboard"**, se recoge la actividad del usuario con el ratón dentro del objeto con la clave **"Mouse".** Aquí se definen las veces que se ha hecho click en **"MouseClicks",** el cálculo de distancia euclídea con las coordenadas del ratón en **"EuclideanDistance"** y las veces que se ha cambiado de dirección con el scroll del ratón en **"MouseWheel".**

Cada vez que se recoge la actividad del usuario, pasado el tiempo que se ha definido previamente, se añadirá un objeto con el **"Time", "Keyboard"** y **"Mouse"** con la nueva actividad recogida.

En la siguiente figura se muestra un ejemplo con la estructura que posee el fichero:

```
{
  "TimeActivityPerSecond": "10",
  "MouseSamplePerSecond": "10",
  "Activity": [
    {
      "Time": {
        "TimeElapsed": "26/06/2020 21:12:38"
      },
      "Keyboard": {
        "PressedKeys": 0,
        "BackSpace": 0,
        "TwoPressedKeys": 0
      },
      "Mouse": {
        "MouseClicks": 6,
        "EuclideanDistance": 10917.458645385177,
        "MouseWheel": 0
      }
    }]
}
```


Figura 3.5 Ejemplo de fichero de actividad

El fichero, mientras está escribiendo la actividad del usuario, también es utilizado para la creación de los gráficos radiales que se explicarán en el capítulo 3.2.1.4.1.

3.2.1.3.2. WordsFile.cs

En la clase **`WordsFile.cs`** se define el fichero .txt que contiene todas las palabras que va escribiendo el usuario. Cada vez que se detecta que el usuario ha pulsado la tecla “enter” o ha pulsado el click izquierdo del ratón, se interpreta como una nueva línea, para así diferenciar que cada línea del fichero es como si fuera una frase.

Este fichero es utilizado por el script de Python del análisis de la clasificación de textos por emociones en un hilo de ejecución en paralelo. Debido a esto, si el script y el programa actual utilizaban el mismo fichero a la vez ocurría un error que decía que había otro proceso que estaba utilizando ese fichero. Por ello, para solucionar este problema, se creó un segundo fichero que contiene lo mismo pero que se cierra cada vez que se escribe en él. Esto lo que permite es que uno de los ficheros se utilice para el programa principal y otro para el script.

3.2.1.4. Clasificación del comportamiento del usuario

Para clasificar el comportamiento del usuario y distinguir cuándo está usando el ordenador de una forma anómala y cuándo no, lo primero que se hizo fue analizar cómo usaba el usuario el ordenador de forma “normal”. Para obtener y distinguir cuando se está usando el ordenador de forma “normal”, se utilizó el fichero de actividad y se analizó y limitó que los valores obtenidos en el fichero serían los de “estado de normalidad”, y por ello, si se salía de esos valores se consideraría que el usuario está haciendo un uso anormal.

Para regular los valores que se consideran normales y los que no, se utiliza la clase **`AlarmForm.cs`**.

3.2.1.4.1 AlarmForm.cs

`AlarmForm.cs` es la clase que sirve como formulario para determinar los valores que se considerarán anómalos en cada uno de los parámetros que se están midiendo. Este formulario actualiza su información cada 11 segundos para así ver, en el momento, la actividad actual del usuario. En la siguiente figura, se puede observar cómo sería el formulario creado con esta clase.

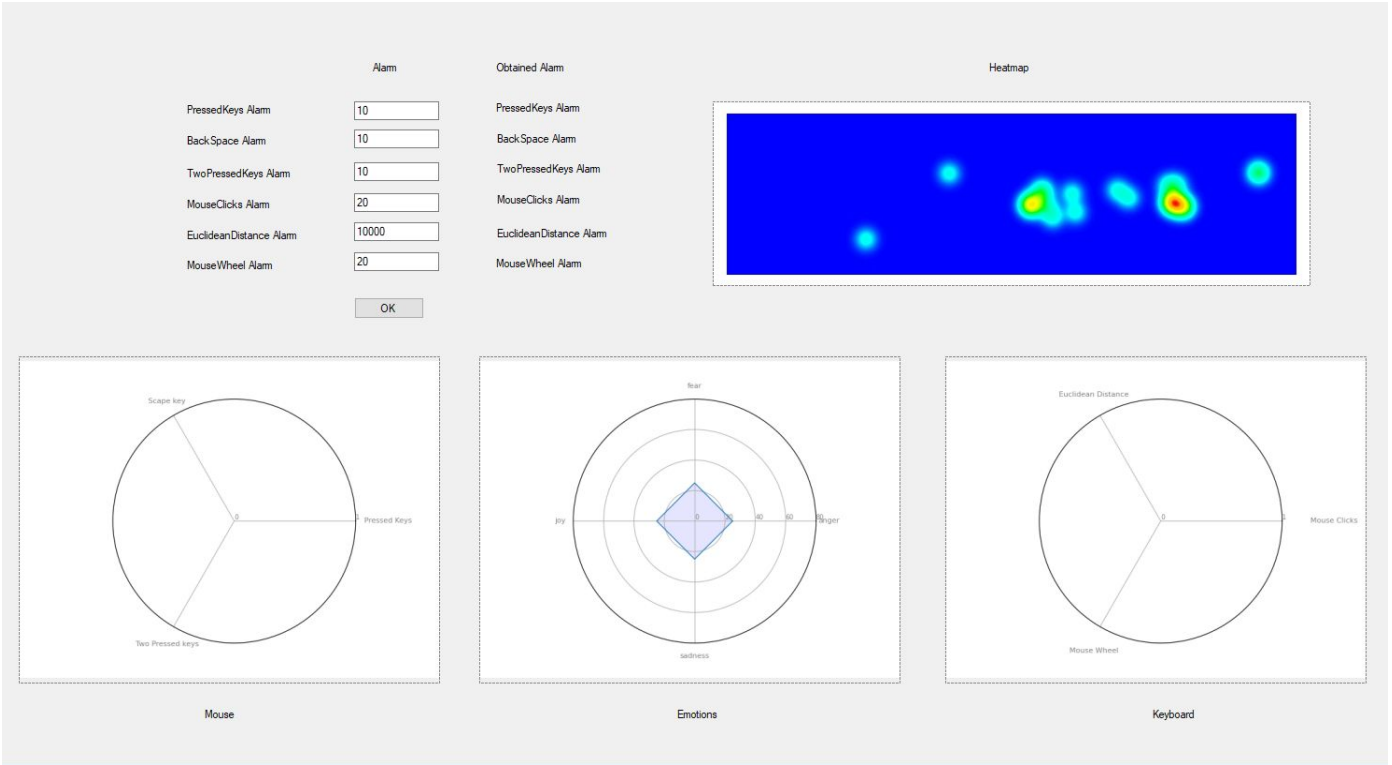

Figura 3.6 Formulario con actividad del usuario

El formulario, se divide en varias secciones: las alarmas que definen los límites entre los que se encuentra una actividad “normal”, el mapa de calor generado

gracias a la actividad del usuario con el ratón por la pantalla y los gráficos con la información obtenida del teclado, del ratón y de la clasificación por emociones.

Tal como se observa en la figura siguiente, en esta sección del formulario se encuentran todos parámetros que están midiendo del comportamiento del usuario.

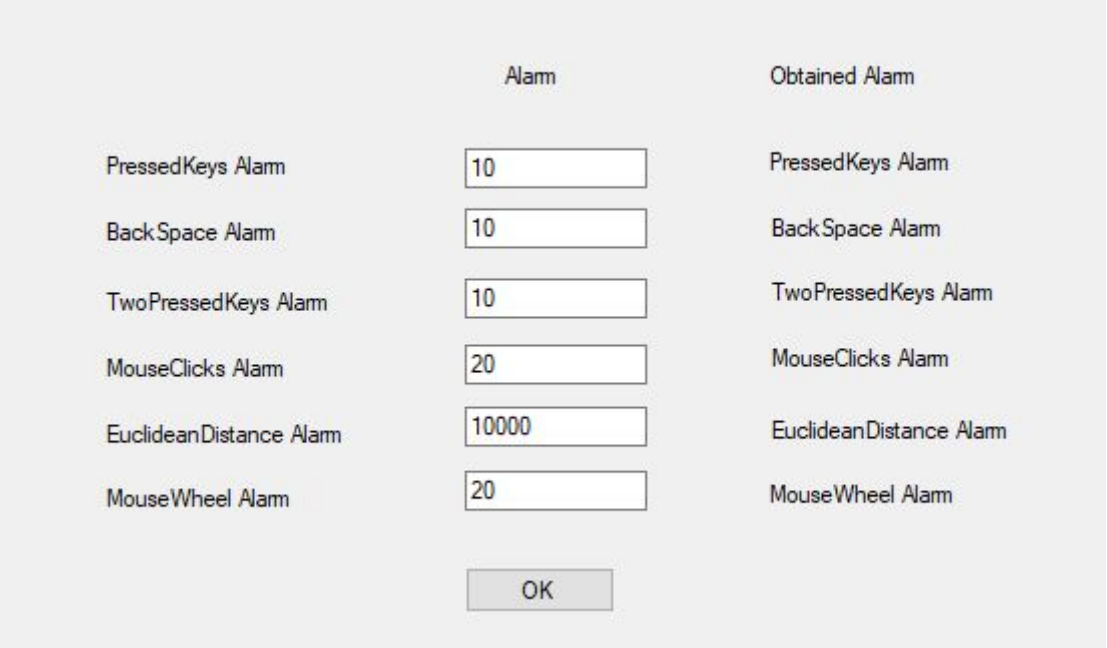

Figura 3.7 Sección del formulario con las alarmas de la actividad del usuario

La primera columna indica el nombre de los parámetro, los cuáles son:

Nombre en la fila	Medida a la que hace referencia
PressedKey Alarm	Pulsación de teclas en un tiempo determinado.
BackSpace Alarm	Nº de veces que se pulsa retroceder en un tiempo determinado.
TwoPressedKey Alarm	Nº de veces que se pulsan dos teclas simultáneamente en un tiempo determinado.
MouseClicks Alarm	Pulsación de clicks del ratón en un tiempo determinado.
EuclideanDistance Alarm	Cálculo de distancia euclídea en un tiempo determinado.
MouseWheel Alarm	Medir el cambio de dirección del scroll en un tiempo determinado.

Tabla 3.3 Alarmas

En la segunda columna es donde se indica a partir de qué valor se considera anómala o no dicha medida. Este valor puede ser cambiado en cualquier momento

simplemente escribiendo otro valor en la fila que se quiera cambiar, y luego pulsando en el botón de “OK” para que se apliquen los cambios.

Por último, en la tercera columna, se indica el valor que se ha obtenido de la medida tras 11 segundos de ejecución. Si el valor de la medida es mayor que el que se indica en la segunda columna, éste cambiará de color a rojo, en cambio, si no supera el valor aparecerá en verde e indicará que está dentro de los valores normales. Cuando la medida se sale de los valores normales establecidos, el usuario recibirá una notificación o pop-up de aviso. Esto sirve para que cuando el formulario esté cerrado, se pueda saber en cada momento si se está produciendo alguna anomalía.

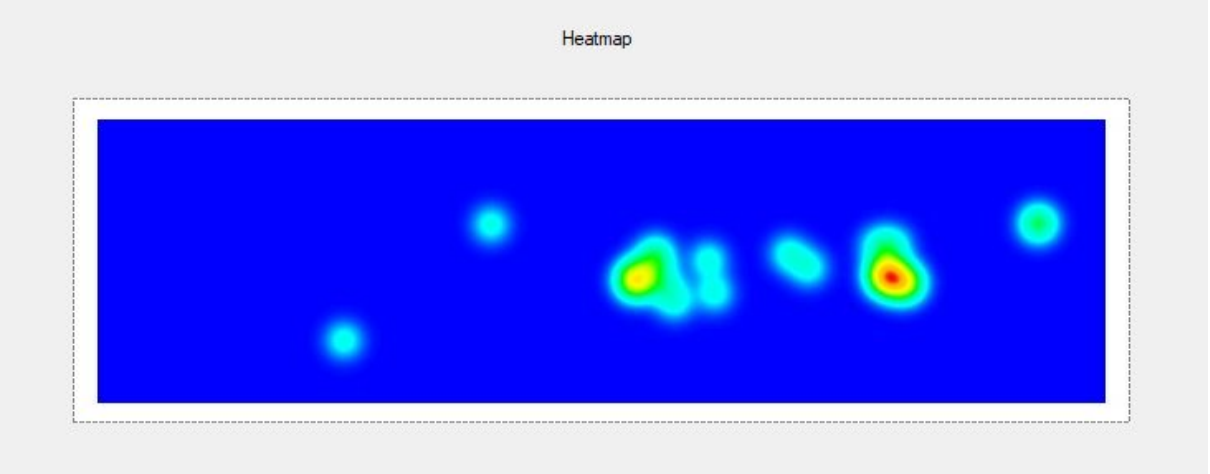

Figura 3.8 Sección del formulario con el mapa de calor generado por el usuario

En la sección del formulario que se muestra en la anterior figura, aparece un mapa de calor que indica por dónde se ha movido el usuario más frecuentemente. Este gráfico se realiza a través del script que se explicará en el capítulo 3.5. y se muestra en el formulario, gracias a un objeto de C# para la creación de formularios **`PictureBox`** que permite que se muestren imágenes. Cada 11 segundos, se actualizará con la nueva información recogida y se mostrará la nueva imagen generada.

Figura 3.9 Sección del formulario con el mapa de calor generado por el usuario

En la parte inferior del formulario, se encuentran los gráficos que se muestran en la figura anterior que representan la información del teclado, del ratón y del clasificador por emociones.

Los gráficos de “Keyboard” y “Mouse” representan las medias de los parámetros que se van guardando en el fichero de actividad. Cada vez que el usuario se sale de los valores que se consideran normales, se pinta una línea roja indicando que se ha obtenido un 1 en esa medida, en cambio si la medida no se ha salido de los valores se quedará a cero. Para poder establecer que los valores normales se representarán con un “0” y los valores anormales se representarán con un “1”, el formulario le pasa al script de creación de los gráficos, un archivo con los valores que se han establecido en las alarmas. El funcionamiento y la creación de los gráficos se detallará más en profundidad en el capítulo 3.3.

Por otro lado, el gráfico de “Emotions” representa los valores que se obtienen cuando se realiza la clasificación de las frases que escribe el usuario por emociones. Cada uno de los extremos indica una emoción, por lo que, dependiendo de las probabilidades que vaya obteniendo el modelo al clasificar las frases, se hará una media y se mostrará en forma de gráfico radial. En este caso, cuando se salga del 40, que indica el 40% de probabilidades obtenidas en la media de las emociones, saltará la “alarma”. El funcionamiento de este gráfico también será explicado más en profundidad en el capítulo 3.4.

3.3. Creación de gráficos radiales con la actividad del teclado y ratón

En esta sección, se va a detallar cómo funciona el script **`graphics_mk.py`** que crea los gráficos radiales con la actividad del teclado y el ratón.

El script consta de las siguientes funciones, las cuales hacen posible la creación de los gráficos radiales:

- **`def json_read(listMean):`** en esta función, se carga el fichero JSON de actividad más reciente en el script. De este modo, la actividad que se muestre en los gráficos será la más actual. Una vez se cargue la información, se

calcula la media de cada parámetro para así sacar las medias de todos ellos y que la información que aparezca en los gráficos sea una media ponderada de la actividad. La media de cada parámetro se guarda en la lista que se le pasa como parámetro a la función.

- **`def json_alarm(listAlarm):`** en esta función, se carga el fichero que contiene los valores a partir de los cuales se consideran anómalos los parámetros que se están midiendo. Por parámetro, se le pasa la lista de alarmas para luego poder utilizarlo en la creación del gráfico.
- **`def set_alarms(listMean,listAlarm,listFinalAlarm):`** en esta función se establece si una medida ha pasado lo que se había considerado como el valor normal o no. Para ello, a la función se le pasa como parámetros la lista con las medias de los parámetros (**`listMean`**) , la lista con los valores para establecer la alarma (**`listAlarm`**) y la lista que se va a rellenar indicando si se ha salido del valor que se consideraba "normal" o no (**`listFinalAlarm`**). Si la media de una medida se ha pasado de lo que se consideraba "normal" se indica con un 1 en la lista **`listFinalAlarm`** para ese valor, mientras que si el valor está dentro de la normalidad se indica con un 0. Al final esta función devuelve la lista formada por unos y ceros, indicando si una medida se ha pasado del valor que se consideraba "normal" o no.
- **`def draw_grafic_keyboard(listFinalAlarm,ax):`** en esta función se crea el gráfico radial con la actividad del teclado. Para ello, se hace uso de la librería pandas [16] que primero crea un Dataframe con la información de las medias de las medidas que contiene la actividad del usuario, para luego ser utilizado en el gráfico.

  Una vez se ha creado, se calcula en una circunferencia los ángulos que tendrá el gráfico radial dependiendo de la información que se vaya a representar. En el caso del teclado será la media de las teclas pulsadas, la media de veces que se pulsa la tecla de retroceso y la media de las veces que se pulsan dos teclas a la vez. Por ello, el gráfico tendrá tres ángulos. Cuando se hayan calculado, se crea un eje con el ángulo establecido mediante la librería matplotlib [12] y se crea el gráfico radial completo con la información. En el gráfico, se mostrará en rojo y como un 1 en ese eje cuando el valor se considere anormal, en cambio cuando esté dentro de la normalidad, se indicará con un 0 y el gráfico no dibujará una línea para ese eje.

  El gráfico creado se guarda como .png en la carpeta del proyecto, para que luego se muestre en el formulario de **`AlarmForm.cs.`**
- **`def draw_grafic_mouse(listFinalAlarm,ax):`** esta función realiza el mismo procedimiento que la explicada anteriormente pero para el caso del ratón. Por ello en este caso, los ejes que se muestran son la media de los clicks del ratón, la media de la distancia euclídea y la media de los cambios

de dirección de la rueda del ratón. En la siguiente figura, se podrá observar un ejemplo del gráfico creado con la actividad del ratón.

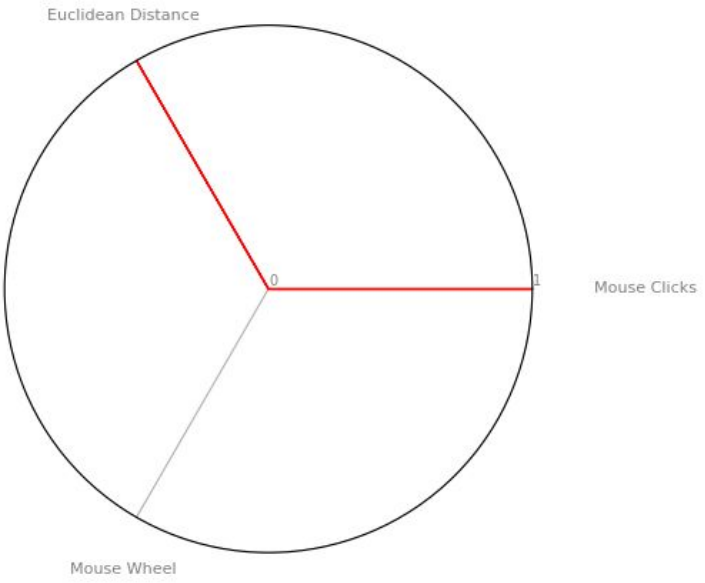

Figura 3.9 Gráfico radial del ratón

- **`def main_alarms(listMean,listAlarm,ax,listFinalAlarm):`** esta función es la que llama a todas las funciones anteriores pasándole la lista de las medias de las medidas (**`listMean`**), la lista con las alarmas (**`listAlarm`**), el objeto que contiene la figura donde se va a trazar el gráfico (**`ax`**) y la lista que indica si los valores de **`listMean`** se han pasado de las alarmas establecidas en **`listAlarm (listFinalAlarm).`**

3.4. Clasificación de frases por emociones

En este capítulo, se va a detallar cómo se realizó el script que clasifica las frases que escribe el usuario a través del teclado por emociones.

3.4.1. Puesta en contexto e investigación

A través de las entradas recogidas por el teclado, el usuario puede expresar su estado de ánimo dependiendo de las frases que escriba y el sentimiento o emoción que ésta evoque en su conjunto. Por ello, una forma de determinar si está sucediendo algo fuera de lo normal, puede ser analizando las frases que escribe el usuario y clasificarlas por emociones. Así, si el usuario escribe muchas frases que evocan tristeza, enfado o miedo se podría decir que algo no está yendo bien,

mientras que, si por el contrario escribe frases que en su conjunto no evocan estas emociones, se podría decir que el usuario está “normal”.

En un primer momento, se investigó cómo realizar una clasificación por sentimientos. De modo que, si se hiciera por sentimientos la clasificación sería indicando si el sentimiento de la frase es positivo, negativo o neutro. Por ello, si el usuario escribe muchas frases que se clasifican como negativas significa que algo no está yendo bien, mientras que, si en su conjunto son positivas o no hay demasiadas negativas, significa que está en un estado de normalidad.

Para realizar una clasificación de texto por sentimientos, se encontraron varias formas de llevarlo a cabo.

Primero se encontró una librería de Machine Learning para C# (ML.NET) [18], la cual te proporciona varios métodos para crear un modelo de clasificación binaria, entrenarlo y luego predecir el sentimiento a partir del modelo creado. A partir del tutorial de la página de microsoft [19], se implementaron todos los métodos que se describen y se probaron para ver cómo funcionaba. El problema fue que el modelo no parecía demasiado preciso y no se podía analizar cómo se obtenían las probabilidades. De hecho, los métodos clasificaban la frase en la clase que tenía la probabilidad más alta pero no se podía ver cómo había deducido esta clasificación. Por ello, se descartó esta implementación y se siguió investigando.

Después, se descubrió cómo realizar un analizador de sentimientos utilizando CoreNLP [20], que es una librería para el procesamiento del lenguaje natural. Con esta librería, sí se podía analizar más en profundidad cómo se realiza la clasificación y las probabilidades obtenidas de que pertenezca a una clase en concreto.De este modo, se decidió realizar un pequeño programa en python que mostraba una capacidad de clasificación muy superior a la de la alternativa anterior pero, en este momento, surgió la siguiente cuestión: ¿clasificar las frases del usuario en sentimientos positivos o negativos nos permite realmente saber si se está produciendo una anomalía o no?. Cuando se usaba el modelo para clasificar frases en positivas o negativas, éste solo los clasificaba mirando si el sentido de la frase era negativo o positivo, por lo que, si el usuario escribía frases como “quiero morirme”, el clasificador diría que es una frase positiva ya que no mide la positividad del sentimiento sino si tiene connotaciones positivas o negativas. Por ejemplo, si se escribe “la película es horrible”, en este caso el clasificador sí diría que es negativa, ya que la frase tiene connotaciones negativas. Este tipo de clasificadores son adecuados para analizar las opiniones de películas o sobre un restaurante, ya que en este caso sí interesa saber si la frase tiene connotaciones positivas o negativas, pero en el caso de esta aplicación lo que se pretendía saber es si el usuario tiene un sentimiento que le hace sentir mal o realmente se siente bien y positivo. Por ello, para poder analizar estas sensaciones se decidió que la clasificación de las frases fuera por emociones.

Cuando se empezó a investigar cómo realizar el modelo para clasificar frases

por emociones, se encontró una librería en Python llamada Scikit-learn, la cual se explica más detalladamente en el capítulo 2.1.1.2., que poseía varios modelos de clasificación y varios métodos que permitían realizar lo que se estaba buscando de manera rápida y sencilla, y por eso, fue el elegido para la implementación.

3.4.2. Estructura del código

Para la creación del clasificador de textos por emociones, primero se creó el script **`comparation_algorithms.py`** en el que se comparan varios algoritmos de machine learning para la creación de modelos de clasificación y se escoge, entre todos ellos, el que tenga más precisión a la hora realizar la clasificación. Una vez obtenido el mejor modelo, éste se utiliza en el script **`graphics_emotions.py`** para clasificar dinámicamente las frases que vaya escribiendo el usuario por teclado, y así, poder crear un gráfico radial con la media de las emociones que está experimentando el usuario. En la siguiente figura, se muestra un diagrama con el flujo que sigue el clasificador por emociones.

Figura 3.10 Diagrama del flujo de código del clasificador por emociones

3.4.3. Pasos para crear el clasificador de textos por emociones usando machine learning

El Machine Learning o aprendizaje automático es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente a partir de datos. Para la creación del clasificador de textos por emociones se han usado técnicas de esta disciplina que crean modelos que resuelvan la tarea en cuestión. En la creación de modelos de Machine Learning se suele seguir una serie de pasos, que en en el caso del clasificador de textos por emociones han sido los siguientes:

1. **Obtener un dataset**: el primer paso es obtener un dataset o conjunto de datos que, posteriormente, se pueda utilizar para la creación del modelo.
2. **Procesar el dataset:** a continuación, se transforma el dataset para facilitar la tarea de clasificación.
3. **Dividir el dataset en el conjunto de testeo y el de entrenamiento:** a continuación, se realiza una división de los datos en un subconjunto de entrenamiento y testeo para que luego, sean utilizados para entrenar y

probar el modelo. El cómo dividir estos conjuntos afectará directamente a la precisión del modelo.

4. **Entrenar distintos modelos con varios algoritmos de clasificación:** en este paso, se entrenarán varios modelos con diferentes algoritmos para comparar las precisiones obtenidas con cada uno de ellos. El que tenga la mayor precisión, será el elegido para resolver el problema de la clasificación de los textos por emociones.
5. **Evaluación:** una vez entrenado el modelo, se probará con los datos de prueba para así verificar la eficacia del modelo.
6. **Usar el modelo para hacer predicciones:** una vez obtenido el modelo, se podrá utilizar para realizar predicciones.

Todos estos pasos, están implementados en el script de **comparation_algorithms.py**, para así obtener el mejor modelo de clasificación.

3.4.3.1. Obtención del dataset

Para crear el modelo de clasificación de textos por emociones, se necesitaba encontrar un dataset o conjunto de datos con el que probar y entrenar el modelo.

Primero se intentó buscar datasets en español, pero como no se encontró ninguno que se adaptara a lo que se estaba buscando, se decidió buscarlos en inglés y, antes de hacer la clasificación del texto, usar googletrans [15] para traducir la frase al inglés.

Tras buscar varios datasets en inglés para clasificar textos por emociones, se encontró y escogió el dataset del proyecto ISEAR [21] (International Survey on Emotion Antecedents and Reactions).

El proyecto ISEAR surgió durante la década de 1990, dirigido por Klaus R. Scherer y Harald Wallbott, en el cual un gran grupo de psicólogos de todo el mundo recolectó datos sobre encuestas que se le hacía a varios estudiantes y psicólogos. En ellas, se les pidió que informaran sobre situaciones en las que habían experimentado las 7 emociones principales (alegría o normal, miedo, ira, tristeza, asco, vergüenza y culpa). Esto dio lugar a un conjunto de datos que contenía informes sobre las siete emociones de unos 3000 encuestados de alrededor de personas de 37 países diferentes.

El dataset viene en formato .csv y contiene 7665 frases clasificadas por las siguientes emociones:

Emoción	Número de frases del dataset
Alegría o normal	1094
Miedo	1095
Ira	1096
Tristeza	1095
Asco	1096
Vergüenza	1096
Culpa	1093

Tabla 3.4 Número de frases del dataset ISEAR

Aparte de escoger el dataset de ISEAR, también se utilizó un dataset encontrado en un proyecto de github [22], el cual contiene frases clasificadas por emociones al igual que el de ISEAR. Este dataset también viene en formato .csv y contiene 7652 frases clasificadas por las siguientes emociones:

Emoción	Número de frases del dataset
Alegría o normal	1092
Miedo	1093
Ira	1094
Tristeza	1094
Asco	1094
Vergüenza	1094
Culpa	1091

Tabla 3.5 Número de frases del dataset de github

Para tener un mayor conjunto de datos, lo que se hizo fue concatenar ambos dataset. De este modo, el dataset aumentó su número de frases a 15.317.

De todas las emociones que había en el dataset, solo se escogieron las emociones de miedo, alegría, ira y tristeza, ya que son las emociones más relevantes para lo que se quiere tratar en este caso.

Una vez obtenido el dataset, lo primero que se realizó es cargar el conjunto de datos en el script. La carga de esta información, se realiza como se muestra en la siguiente figura.

```
data=pd.read_csv('isear.csv',sep=';')
data=data[['sentiment','content']].copy()
```

Figura 3.11 Código para cargar dataset

En estas dos líneas lo que se hace es que se lee el fichero .csv cuyo separador es ”;”, y después, se indica que la información viene en dos columnas, una que indica el sentimiento y otra que contiene el contenido del texto.

3.4.3.1.1. Procesamiento del dataset

El procesamiento del dataset antes de utilizarlo en la creación del clasificador es muy importante, ya que como el clasificador aprende a partir de él, el cómo estén los datos afecta directamente en el aprendizaje y en la precisión del mismo.

Para realizar el procesamiento del dataset se aplicaron las siguientes técnicas:

- **Tokenización (eliminar caracteres irrelevantes):** este proceso consiste en realizar una limpieza y transformar las palabras, de tal manera que se eliminan todos los caracteres irrelevantes y solo se cogen los caracteres alfanuméricos y los espacios.
  En el script, esto se realiza mediante expresiones regulares, reemplazando por nada todo lo que no cumpla lo descrito anteriormente, tal y como se muestra en la siguiente figura.

```
data['content']=data['content'].str.replace('[^A-Za-z0-9\s]+', '')
```

Figura 3.12 Línea de código para hacer tokenización

- **Convertir todo a minúsculas:** en este proceso se convierten todas las palabras a minúsculas.
  En el script se realiza de la forma que se muestra en la siguiente figura.

```
data['content']=data['content'].str.lower()
```

Figura 3.13 Línea de código para convertir todo el dataset en minúscula.

- **Lematización:** con este proceso se cambian todas las palabras por su lema. El lema de una palabra es la palabra que nos encontraríamos como entrada en un diccionario tradicional: singular para sustantivos, masculino singular para adjetivos, infinitivo para verbos. Gracias a este proceso, se pueden obviar las diferencias entre las palabras que significan lo mismo pero tienen diferentes variantes, y juntarlas todas en un mismo término.
  Para realizar la lematización, se utilizó el corpus que provee la librería de nltk [14] para buscar si la palabra era un sustantivo, un adjetivo o un verbo, y cuando se detecte qué tipo de palabra es, buscar su lema equivalente y sustituirla en el dataset.
- **Stemming:** este proceso consiste en reducir una palabra a su raíz.
  El stemming se realiza de la misma forma que la lematización en el script pero en vez de utilizar el corpus para buscar qué tipo de palabra es, se utiliza el módulo **`nltk.stem`**, también de la librería nltk, para reducir cualquier palabra a su raíz.
- **Quitar los stop words o palabras vacías:** éstas son las palabras que carecen de un significado por sí solas, y por eso, es mejor eliminarlas ya que solo producen ruido al ser utilizadas en el entrenamiento del clasificador. Las palabras vacías suelen ser artículos, preposiciones, conjunciones y pronombres.
  Para quitar los stopwords en el script se detecta primero los stopwords en inglés y luego se buscan para ser eliminados del contenido. En la siguiente figura se muestra cómo se realizó esta operación en el script.

```
stop_words = set(stopwords.words('english'))
data['content'] = [w for w in data['content'] if not w in stop_words]
```

Figura 3.14 Línea de código para quitar los stop words o palabras vacías.

3.4.3.2. División del conjunto de entrenamiento y de prueba

Antes de entrenar un modelo, hay que hacer una división del dataset, en un subconjunto para entrenar un modelo y un subconjunto para probar el modelo entrenado. De este paso depende, en gran medida, la precisión del modelo.

```
X_train, X_test, y_train, y_test =
train_test_split(data,target,stratify=target,test_size=0.1,
random_state=42)
```

Figura 3.15 Código para hacer la división del conjunto de entrenamiento y de prueba.

Para dividir el dataset, se ha usado la función **`train_test_split`** de la forma que se muestra en la figura anterior. Todos los parámetros de la función tienen un significado que se detallará a continuación:

- **`data, target:`** Es el conjunto de datos que se va utilizar para realizar la división.
- **`stratify=target:`** Indica que se va a utilizar la **`StratifiedShuffleSplit`** función para realizar la división del dataset. Esto lo que hace, es que divide preservando el porcentaje de muestras para cada clase.
- **`test_size:`** Este parámetro especifica el tamaño del conjunto de datos de prueba. En este caso, el conjunto de prueba será un 10% del total del dataset y el resto será para el conjunto de entrenamiento.
- **`random_state:`** indicando este parámetro, se realiza una división aleatoria usando np.random.

Una vez realizada la división del dataset, se obtendrán los siguientes datos:

- **`X_train`** con los datos para entrenar.
- **`y_train`** con las “etiquetas” de los resultados esperados de **`X_train.`**
- **`X_test`** con los datos para test.
- **`y_test`** con las “etiquetas” de los resultados de **`X_test.`**

Después de realizar la división y de obtener los datos divididos, se transformaron dichos datos en vectores de recuento de términos / tokens con el módulo **`CountVectorizer`** de la librería scikit-learn [13] como se muestra en la siguiente figura.

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train.content)
X_test_counts = count_vect.transform(X_test.content)
```

Figura 3.16 Código para crear el objeto de la clase CountVectorizer

Una vez transformados los datos, ya se pueden usar para la creación del modelo de clasificación.

3.4.3.3 Algoritmos de clasificación para crear un modelo

Un modelo es lo que surge tras entrenar un sistema con un algoritmo, es decir tras detectar los patrones en los datos, para posteriormente realizar predicciones. Existen muchos algoritmos con los que se puede entrenar un clasificador de textos por emociones, pero en este caso se ha elegido poner a prueba varios, para así quedarnos con el que tenga mayor precisión.

Para poner a prueba todos los algoritmos que se van a detallar a continuación, se creó una función con la que se hace una prueba de rendimiento (benchmark) para medir cómo sería la precisión, el tiempo que tarda en el entrenamiento y el tiempo que tarda en el testeo el modelo creado a partir del algoritmo en cuestión.

La función creada para atender esta tarea se llama **`benchmark(clf)`** y, para poder poner a prueba todos los algoritmos, se le pasa por parámetro el algoritmo que se esté utilizando en cada momento de la prueba de rendimiento. Dentro de la función, se entrena el modelo con el conjunto de entrenamiento y luego se evalúa con el conjunto de testeo. Mientras esto ocurre, se va midiendo lo que tarda en crear el modelo en su fase de entrenamiento y evaluación. Una vez se termina de crear el modelo, se obtiene la precisión calculando el ratio entre las predicciones correctas (suma de verdaderos positivos y verdaderos negativos) y las predicciones totales.

La información del tiempo de entrenamiento, el tiempo de testeo y la precisión se guarda en una lista para así luego poder comparar todos estos parámetros y, de este modo, elegir el mejor modelo.

La función detallada anteriormente, se muestra en la siguiente figura.

```
def benchmark(clf):
    print('_' * 80)
    print("Training: ")
    print(clf)
    t0 = time()
    model.append(clf.fit(X_train_counts,y_train))
    train_time = time() - t0
    training_time.append(train_time)
    print("train time: %0.3fs" % train_time)

    t0 = time()
    pred = clf.predict(X_test_counts)
    test_time = time() - t0
    test_times.append(test_time)
    print("test time:  %0.3fs" % test_time)

    score = metrics.accuracy_score(y_test, pred)
    scores.append(score)
    print("accuracy:   %0.3f" % score)
```

Figura 3.17 Código de la función benchmark

Todos los algoritmos que se van a testear en el script provienen de la librería scikit.learn [13].

3.4.3.3.1. KNN

El **K-Nearest-Neighbor (KNN)** es un algoritmo basado en instancia, es decir, que crea un modelo a partir de una base de datos y se agregan nuevos datos comparando su similitud con las muestras ya existentes, para así, encontrar los mejores k vecinos y hacer la predicción.

El funcionamiento del algoritmo es el siguiente:

1. Calcula la distancia entre el elemento a clasificar y el resto del dataset de entrenamiento.
2. Selecciona los “k” ítems con menor distancia utilizando la distancia euclídea o la cosine similarity.
3. Realiza una “votación de mayoría” entre los k puntos. Esto significa que los de una clase/etiqueta que dominen decidirán su clasificación final.

Para poner a prueba este algoritmo en el script, se hace uso del módulo **`KNeighborsClassifier`** de la librería scikit.learn [13].

3.4.3.3.2. Logistic Regression

La **regresión logística (Logistic Regression)** es un algoritmo que proviene del campo de la estadística y que su principal aplicación es en problemas de

clasificación, en los que se obtiene valores binarios, entre 0 y 1. Con este algoritmo, se mide la relación entre la variable dependiente, la afirmación que se desea predecir, con una o más variables independientes, el conjunto de características disponibles para el modelo.

Para calcular la probabilidad de la variable dependiente, se utiliza una función logística o función sigmoidea, la cual es una curva en forma de S que puede tomar cualquier valor siempre que esté entre los valores 0 y el 1. La ecuación que se aplica para la función sigmoidea es:

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

Figura 3.18 Ecuación de la función sigmodea

Esta ecuación genera gráficas como la que se muestra a continuación:

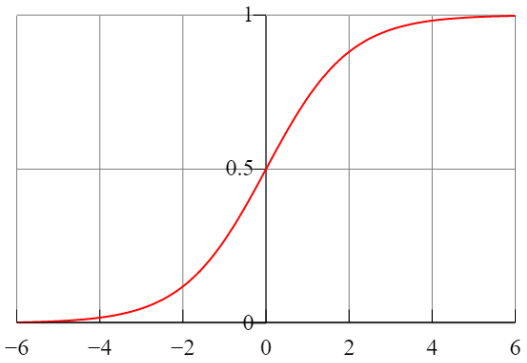

Figura 3.19 Función sigmodea

En este caso, el módulo importado para usar este algoritmo ha sido el **`LogisticRegression.`**

3.4.3.3.3. Nearest Centroid

Nearest Centroid o centroide más cercano es un algoritmo que se usa en los modelos de clasificación que asigna a las observaciones de la etiqueta de la clase de muestras de entrenamiento cuyo centroide (media) es la más cercana a la observación. Cuando se aplica a la clasificación de texto, como es en este caso, el clasificador centroide más cercano es conocido como el clasificador Rocchio ya que es muy similar al algoritmo de Rocchio.

El algoritmo funciona de tal forma que con el conjunto de entrenamiento se calculan los centroides por clase y luego se aplica la función de predicción, midiendo las distancias euclídeas entre el punto que se está intentando predecir y los

centroides de las clases. Tras aplicar la función de predicción, el que haya dado la distancia más corta entre el punto de datos que se está intentando predecir y el centroide de la clase será el que determine a qué clase pertenece.

Para poner a prueba este algoritmo en el script, se hace uso del módulo **`NearestCentroid`**.

3.4.3.3.4. Naive Bayes

Naive Bayes es uno de los modelos probabilísticos más usados para la clasificación de textos. Se basa en la aplicación del Teorema de Bayes, también conocida por el Teorema de la probabilidad condicionada cuya fórmula es la siguiente:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)} = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{n} P(B|A_j)P(A_j)}$$

Figura 3.20 Fórmula del Teorema de Bayes

Donde:

P(Ai|B): es la probabilidad del suceso Ai dado los datos del suceso B, conocido como las probabilidades a posteriori.

P(B|Ai): es la probabilidad de B en la hipótesis de Ai.

P(Ai): es la probabilidad de que el suceso Ai sea cierto (independientemente de los datos), conocido como las probabilidades a priori.

P(B): probabilidad de los datos (independientemente del suceso B).

Dada que la probabilidad de los datos P(B) no aporta información para la clasificación, el término suele omitirse. Por ello finalmente, la probabilidad de un suceso dada la clase se suele calcular como la probabilidad conjunta de cada uno de los sucesos dentro de la clase.

A partir del modelo de Naive Bayes, surgieron variantes al modelo como el **`Multinomial Naive Bayes`** o el **`Complement Naive Bayes`**. Estos modelos son los que se usarán para probar su precisión en la clasificación de textos por emociones.

3.4.3.3.4.1. MultinomialNB

En este modelo, la particularidad que tiene con respecto al Naive Bayes es que se considera la frecuencia de aparición de cada término en vez de la ocurrencia binaria.

Para utilizar este algoritmo en el script, el módulo correspondiente de la librería scikit-learn, es el **`MultinomialNB.`** Para usarlo, simplemente se tiene que llamar a la función **`MultinomialNB()`**.

3.4.3.3.4.2. ComplementNB

Este modelo es una adaptación del **`MultinomialNB`** que se utiliza muy bien para datos que no están del todo balanceados.

Para **`ComplementNB`** el módulo utilizado en el script para llamar al algoritmo es el **`ComplementNB.`**

3.4.3.3.5. SVC

SVC (Clasificación de vectores de soporte) se trata de un algoritmo de aprendizaje supervisado que dado los datos de entrenamientos etiquetados, genera un hiperplano óptimo que categoriza nuevos ejemplos. En los espacios dimensionales, este hiperplano es una línea que divide un plano en dos partes donde en cada clase se encuentra a cada lado.

Para utilizar el algoritmo SVC en el script, hay que importar el módulo **`SVC.`**

3.4.3.3.6. LinearSVC

LinearSVC (Clasificación de vectores de soporte lineal) es un algoritmo similar al anterior, pero en este caso a la hora de ser implementado se añade el parámetro **`kernel="linear"`**, que lo que permite es que tenga más flexibilidad en la elección de penalizaciones y funciones de pérdida, por lo que debería escalar mejor a grandes cantidades de muestras.

Al igual que los anteriores algoritmos, para utilizar este algoritmo en el script, hay que importar el módulo `LinearSVC` de la librería scikit-learn.

3.4.3.4. Entrenar el modelo y evaluación

Para entrenar un modelo, se utiliza la función común que poseen todos los algoritmos, **`fit(X_train_counts,y_train)`**. Con dicha función, se crea el modelo a partir de los datos de entrenamiento que se le pasan por parámetro a la función. De esta forma, se obtendrá el modelo entrenado y con él se podrán realizar predicciones.

Por otro lado, para la evaluación y obtención de precisión del modelo se utiliza la función **`metrics.accuracy_score(y_test, pred)`** pasándole por parámetro el conjunto de etiquetas pronosticado para una muestra y el conjunto de etiquetas correctas.

3.4.3.5. Comparación de las precisiones obtenidas

Una vez obtenidas todas las precisiones, se decidió crear un gráfico de barras utilizando la librería de matplotlib [12], para así ver de forma gráfica cuáles habían sido las precisiones obtenidas.

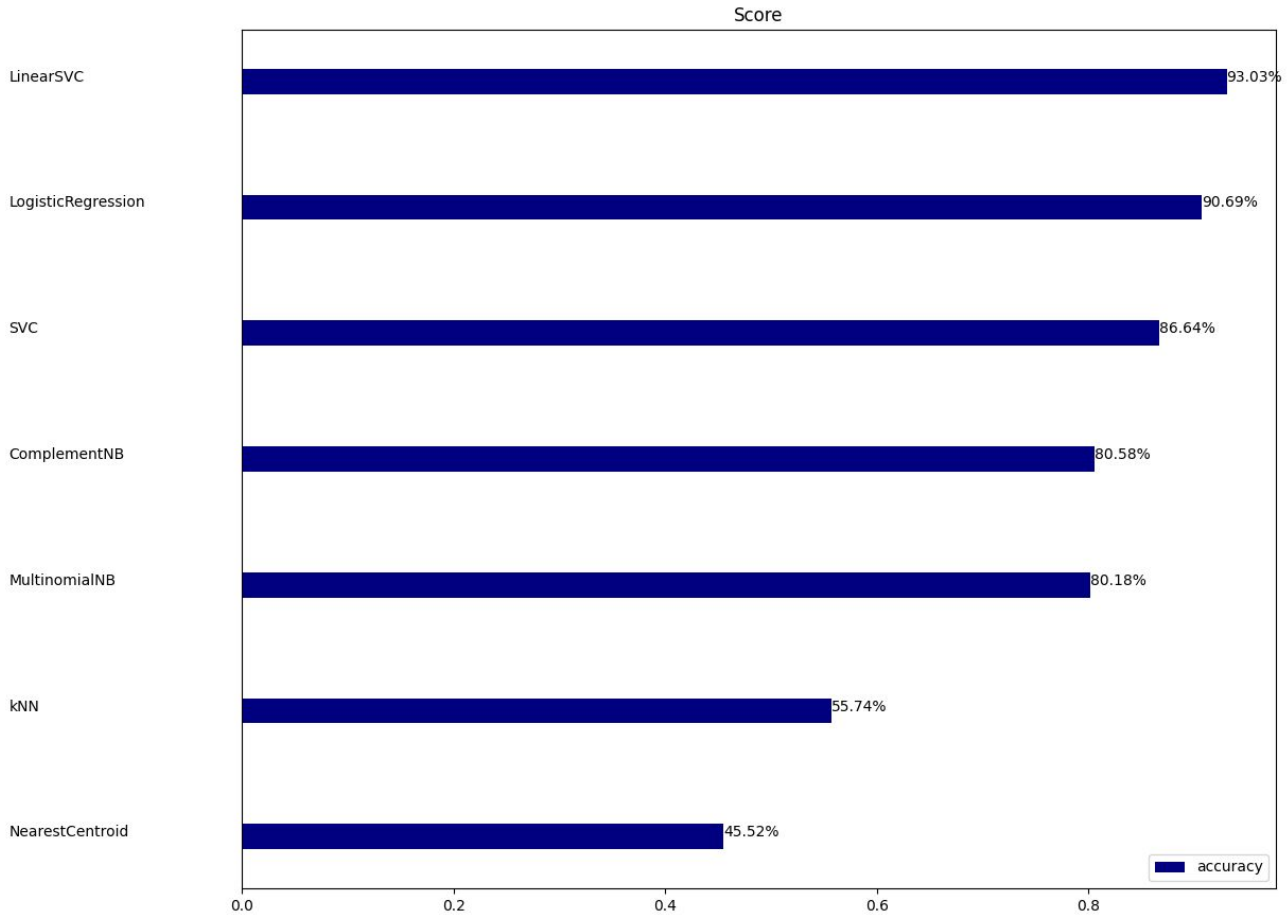

Figura 3.21 Gráfico con las precisiones de los modelos

Tal y como se muestra en la figura anterior, cada algoritmo tiene una barra asociada con la que se indica la precisión del modelo obtenido aplicando dicho algoritmo. Los algoritmos se han ordenados de mayor a menor precisión, por lo que, el modelo que ha utilizado el algoritmo de **LinearSVC** es el que ha obtenido la precisión más alta (93,03%), y por tanto, es el que se escogerá para realizar la clasificación de textos por emociones.

Una vez se ha determinado el modelo con mayor precisión, éste se guarda usando la librería pickle, de la forma que se muestra en la siguiente figura, para así poder utilizarlo sin tener que volver a entrenarlo de nuevo.

```
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))
pickle.dump(count_vect, open("vectorizer.pickle", "wb"))
```

Figura 3.22 Código de la carga del modelo utilizando pickle

Como se puede observar, también se guarda el vector creado anteriormente ya que es el que se utilizará para obtener las predicciones.

3.4.3.6. Utilización del modelo para hacer predicciones y crear un gráfico radial

Para utilizar el modelo para hacer predicciones con el texto que escribe el usuario por el teclado y crear un gráfico radial, se creó el script **`graphics_emotions.py`**.

Este script está ejecutándose en bucle infinito, hasta que no se cierre la aplicación, esperando y leyendo todas las frases que escribe el usuario por teclado.

Con cada frase leída por el script, se realiza una serie de operaciones antes de clasificarla. Primero se traduce al inglés usando la librería googletrans [15] y después se le hace stemming y lematización a cada palabra, al igual que se hacía con el procesamiento del dataset.

A continuación, se obtienen las probabilidades de pertenecer a cada una de las emociones (ira, miedo,tristeza y alegría) con la función **`predict_proba`**. Esta función devuelve un vector indicando la probabilidad que tiene cada frase de ser clasificada en cada una de las emociones. El orden de las probabilidades indica a qué clase pertenece y la que tenga una probabilidad mayor, será la clase predicha.

En la siguiente figura, se muestra el código de lo que se ha explicado anteriormente.

```
#Ruta del archivo que contiene lo que escribe el usuario
path = os.path.abspath("words2.txt")

#Se abre el archivo, el modelo y el vector para hacer las predicciones
fileToRead = open(path)
loaded_model = pickle.load(open('finalized_model.sav', 'rb'))
vectorizer = pickle.load(open('vectorizer.pickle', 'rb'))

#Se crea la lista con las probabilidades de las emociones
list1 = [0,0,0,0]

#Se crea la figura
ax = plt.subplot(111, polar=True)

'''
Se lee cada frase que escribe el usuario y se va sacando la media de las
probabilidades de cada emoción obtenida, para así, mostrar en el gráfico radial
cómo se siente el usuario de media. Mientras no le llegue ninguna frase al
script se quedará en espera en la última línea que leyó
'''
while 1:
    where = fileToRead.tell()
    line = fileToRead.readline()
    phraseOrigin = line
    phrase = phraseOrigin.lower()
    phrase = translator.translate(phraseOrigin)
    phrase = phrase.text
    phrase = stemming_sentence(phrase)
    phrase = lemmatize_sentence(phrase)
    if not line:
        time.sleep(1)
        fileToRead.seek(where)
    else:
        print(phrase)
        proba = loaded_model.predict_proba(vectorizer.transform([phrase]))
        print(proba)
        f = open("proba.txt", 'a', encoding='utf-8')
        b = '\n'.join(' '.join('%0.2f' %x for x in y) for y in proba)
        for y in proba:
            for x in y:
                num = float('%0.2f' %x)
                list1.append(num*100)
        if phrase != '\n':
            f.write(b)
            f.write('\n')
        f.close()
        draw_graphic(list1,ax)
```

Figura 3.23 Código de la función principal de graphics_emotions.py

Los datos que aparecen en el gráfico radial, serán la media de cada uno de los valores obtenidos en las frases que haya ido clasificando el modelo. Por ello, lo que se mostrará será una media de las emociones que está sintiendo el usuario.

Para realizar el gráfico radial, se definió una función llamada **`draw_graphic(list1,ax)`** a la que se le pasa por parámetro la lista con las medias de las probabilidades de las emociones y la figura para dibujar el gráfico. En la función, se calcula dinámicamente la media de las emociones y luego se dibuja el gráfico al igual que se hizo en el capítulo 3.3. El gráfico resultante se guardará en la carpeta del proyecto para posteriormente ser utilizado por **`AlarmForm.cs`** y que el usuario pueda consultarlo. Este gráfico será algo parecido a lo que se muestra a continuación, con las emociones en los ejes y las probabilidades de cada una de las emociones indicadas mediante los círculos que rodean todo el gráfico.

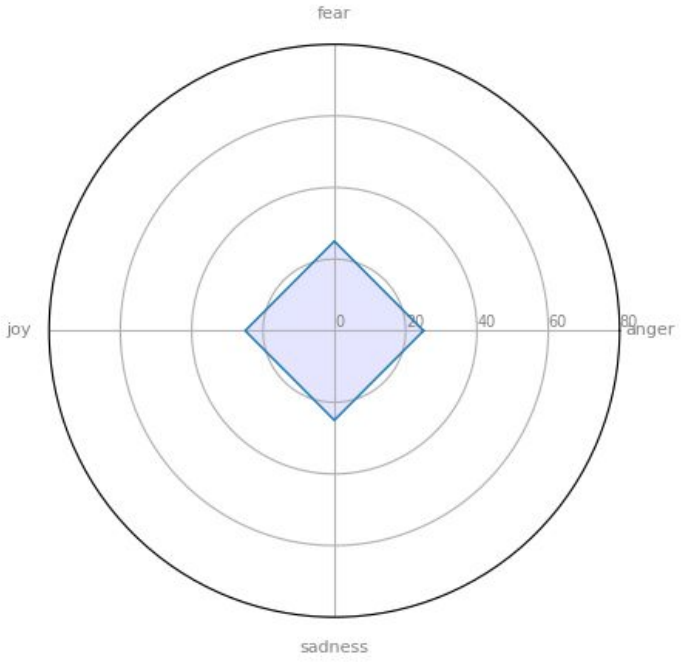

Figura 3.24 Gráfico radial con las emociones

En el caso de que se llegue a más del 45% de las emociones como tristeza, ira o miedo, se considerará que está ocurriendo algo anormal y se avisará al usuario.

3.5. HeatMap

En este capítulo se va a detallar cómo se creó el mapa de calor para mostrar en qué sitios de la pantalla el usuario pasa la mayor parte del tiempo.

3.5.1 Creación del mapa de calor

Un Heatmap o mapa de calor es un gráfico en el que se resaltan mediante un código de colores zonas concretas de una pantalla en base a criterios como el número de clics o las áreas por las que pasa con más frecuencia el puntero. En el caso de este proyecto, se decidió hacer el mapa de calor en base a la posición del ratón en la pantalla.

Para crearlo, se desarrolló un script llamado **`heatmap.py`** en el cual se hace uso de la librería matplotlib [12], al igual que en la creación de los otros gráficos del proyecto, para crear el gráfico que representará el mapa de calor. La particularidad de este script, es que se se usa la clase **`LinearSegmentedColormap`** para crear un mapa de color a partir de segmentos de mapeo lineal.

El script primero capta la posición del ratón y guarda estos datos en una variable que va actualizando conforme el ratón se mueva. Después, se obtiene el tamaño de la pantalla del usuario para así crear la figura que contiene el mapa del mismo tamaño y representar cada punto de la figura como una matriz de ceros. El valor 0 dentro el mapa de colores representa el píxel por el que aún no ha pasado el ratón, por eso en un principio en la matriz todos los valores serán 0. A medida que el ratón se vaya moviendo por la pantalla, se irá aumentando el valor de ese punto en la matriz para que luego se vea reflejado en el mapa con un color más fuerte conforme pase el tiempo. De este modo, dependiendo de los valores de la matriz, se hará un mapeo con una paleta de colores predefinidos y cuanto más fuerte sea el color, significa que ese punto de la matriz tiene un valor más grande por lo que indicará que el usuario ha pasado más tiempo en ese punto con el ratón.

```
colors = [(0, 0, 1), (0, 1, 1), (0, 1, 0.75), (0, 1, 0),
          (0.75, 1, 0), (1, 1, 0),(1, 0.8, 0),(1, 0.7, 0),
          (1, 0, 0)]

cm = LinearSegmentedColormap.from_list('sample', colors)

plt.imshow(data, cmap=cm)
```

Figura 3.25 Código para la creación del mapa de calor

En el trozo de código que se muestra en la figura anterior, se indica cómo se crea el mapa con los colores que se han definido y cómo se mostrará, en base a los datos que se le pasen a la función de creación del mapa.

Los colores que se han definido son los que se muestran en la siguiente figura. Cuando en el mapa se muestran colores parecidos a los que se encuentran más a la izquierda de la figura, significa que en ese punto el ratón no ha permanecido mucho tiempo en la pantalla, y por tanto, los datos correspondientes tendrán valores pequeños. Por el contrario, si se muestran los colores que están más a la derecha, significa que se ha pasado más tiempo con el cursor del ratón en esa zona de la pantalla y en los datos se representarán con valores más grandes.

Figura 3.26 Paleta de colores del mapa de calor

La figura creada, que es parecida a la que se muestra en la figura siguiente, se guardará en la carpeta del proyecto para que así se muestre en **`AlarmForm.cs`**, al igual que los gráficos de teclado, del ratón y de la clasificación por emociones. Cada once segundos, el formulario actualizará el mapa de calor mostrando la información actualizada.

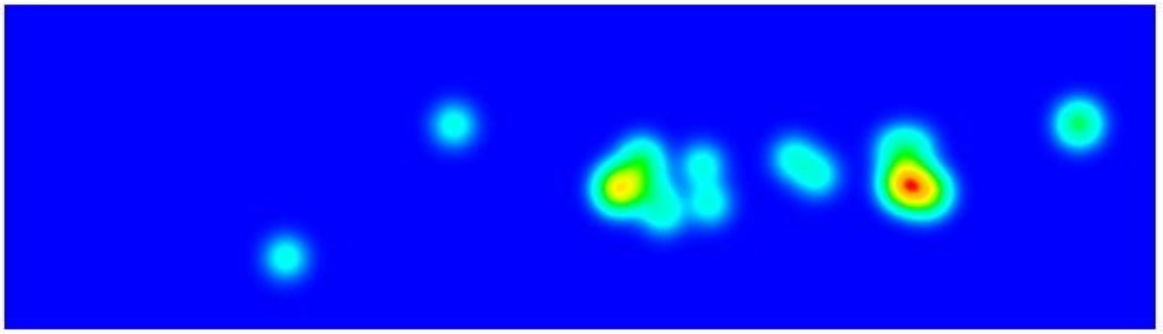

Figura 3.27 Mapa de calor

Capítulo 4. Conclusiones y líneas futuras

El uso de dispositivos móviles y ordenadores está a la orden del día, y esto hace que sea inevitable que la mayor parte del tiempo se estén utilizando. Principalmente, estos dispositivos se emplean para facilitarnos la vida, por ello, con este Trabajo de Fin de Grado, también se ha querido realizar una aportación para ayudar a las personas que puedan sufrir ataques psicóticos y no sean conscientes de ello, pero que gracias a la aplicación, sea más fácil la detección de que algo anómalo está ocurriendo.

En la realización de la aplicación se han adquirido nuevas habilidades y se han afianzado varios conceptos que ya se conocían pero no tan en profundidad. Aunque, aún quedan muchísimas más cosas por aprender de todas las tecnologías utilizadas, ya que ofrecen muchísimas posibilidades de mejora y desarrollo en un futuro.

De esta aplicación, se espera que en un futuro pueda ayudar a las personas que tengan ataques psicóticos y que, a parte de que sea el propio individuo el que vea la alarma de que algo está ocurriendo, que también otra persona pueda recibir alguna especie de notificación y tome las medidas pertinentes.

Otra mejora a tener en cuenta, podría ser la de que la aplicación se entrene de forma automática y no tenga que ser el propio usuario el que, a partir de las medidas que se obtengan después de un rato, tenga que ajustar las alarmas para fijar los límites entre los que se encuentran una situación de normalidad.

Por otra parte, si la aplicación supiera qué actividad está realizando el usuario, ya sea escribir algún texto, leer, jugar, navegar por internet o ver algún vídeo, sería más fácil detectar si realmente está ocurriendo algo anormal, ya que podría ocurrir el caso que esa actividad requiriera salirse de los límites establecidos. Ahora, como está desarrollada la aplicación, se podrían ajustar si se cambia de actividad pero sería más sencillo para el usuario si la aplicación lo detectara por sí solo.

Además de la aplicación de ordenador, también sería interesante poder realizar una versión móvil, ya que este dispositivo se emplea bastante durante la vida cotidiana y podría ser de gran ayuda cuando no se esté utilizando el ordenador.

En cuanto en el desarrollo de la aplicación, se podría llegar a unificar toda la aplicación en un solo lenguaje y hacer uso de algún periférico más para detectar si el usuario realmente está sufriendo un ataque como la cámara o el micrófono.

Por otro lado, con la clasificación de textos por emociones, se podría utilizar para ayudar a personas que tengan trastornos de bipolaridad para detectar si ha cambiado repentinamente de estado de ánimo y avisarlo, al igual que se hace ahora mismo con la aplicación desarrollada.

Capítulo 5. Summary and conclusions

The use of mobile devices and computers is very present nowadays, and this makes it inevitable that most of the time they are being used. Mainly, these devices are used to make life easier, therefore, with this Final Degree Project, we have also wanted to make a contribution to help people who may suffer psychotic attacks and are not aware of it, but thanks to the application, it will be easier to detect that something abnormal is happening.

In carrying out the application, new skills have been acquired and various concepts that were already known, but not so in-depth, have been consolidated. Although, there are still many more things to learn from all the technologies used, since they offer many possibilities for improvement and development in the future.

From this application, is expected to help people who have psychotic attacks and, apart from being the individual himself who sees the alarm that something is happening, that another person may also receive some kind of notification and take appropriate action.

Another improvement to take into account, could be that the application is trained automatically and it does not have to be the user himself who, from the measurements obtained after a while, has to adjust the alarms to set the new limits from what it is considered normal.

On the other hand, if the application knew what activity the user is carrying out, whether it be writing some text, reading, playing games, surfing the Internet or watching a video, it would be easier to detect if something abnormal is actually happening, since it could be the case that this activity requires to exceed the established limits. Now, as the application is developed, we could adjust if it changes its activity but it would be easier for the user if the application detected it on its own.

In addition to the computer application, it could also be interesting to be able to make a mobile version, since this device is used a lot during everyday life and could be very helpful when you are not using the computer.

Regarding the development of the application, it might be considered the possibility to unify the entire application in a single language and make use of some other peripheral to detect if the user is really suffering an attack such as the camera or the microphone.

Alternatively, with the classification of texts by emotions, it could be used to help people who have bipolar disorders to detect if they have suddenly changed their mood and notify them, just as it is done right now with the developed application.

Capítulo 6. Presupuesto

La asignatura Trabajo de Fin de Grado cuenta con 12 créditos y cada crédito equivale a 25 horas, por ello, para el desarrollo de la aplicación se han trabajado 300 horas. Por otro lado, el coste por hora se ha decidido fijarlo en 30€/h. En éste se incluyen amortización de equipos, gastos de suministros e impuestos entre otros. De este modo, el presupuesto quedaría de la siguiente forma:

Descripción	Coste por hora	Total
300 horas de trabajo	30€	9000€

Tabla 6.1 Presupuesto

Bibliografía

[0] “Esquizofrenia.” *World Health Organization*, World Health Organization, www.who.int/es/news-room/fact-sheets/detail/schizophrenia.

[1] Domínguez, Ignacio X., et al. “Detecting Abnormal User Behavior through Pattern-Mining Input Device Analytics.” *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security - HotSoS '15*, 2015, doi:10.1145/2746194.2746205.

[2] Inc, Fireworld. “Responsabilidad.” *Fireworld*, fireworld-supervision.es/responsibility.htm.

[3] “Software Gratis De Escritorio Remoto.” *AeroAdmin*, www.aeroadmin.com/es/.

[4] Kwhat. “Kwhat/Jnativehook.” *GitHub*, github.com/kwhat/jnativehook.

[5] “Schizophrenia and Related Disorders Alliance of America.” *Sardaa*, sardaa.org/.

[6] BillWagner. “Introducción Al Lenguaje C# y .NET Framework.” *Introducción Al Lenguaje C# y .NET Framework | Microsoft Docs*, docs.microsoft.com/es-es/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework.

[7] “.NET Framework 4.5.” *Microsoft*, www.microsoft.com/es-es/download/details.aspx?id=30653.

[8] Gmamaladze. “Gmamaladze/Globalmousekeyhook.” *GitHub*, github.com/gmamaladze/globalmousekeyhook.

[9] “Introducción a JSON.” *JSON*, www.json.org/json-es.html.

[10] 12.0.3, Version. “Json.NET.” *Newtonsoft*, www.newtonsoft.com/json.

[11] “Welcome to Python.org.” *Python.org*, www.python.org/.

[12] “Visualization with Python.” *Matplotlib*, matplotlib.org/.

[13] “Learn.” *Scikit*, scikit-learn.org/stable/.

[14] “Natural Language Toolkit.” *Natural Language Toolkit - NLTK 3.5 Documentation*, www.nltk.org/.

[15] “Googletrans.” *PyPI*, pypi.org/project/googletrans/.

[16] *Pandas*, pandas.pydata.org/.

[17] “Pickle - Python Object Serialization.” *Pickle - Python Object Serialization - Python 3.8.3 Documentation*, docs.python.org/3/library/pickle.html.

[18] “ML.NET: Machine Learning Made for .NET.” *Microsoft*, dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet.

[19] Luisquintanilla. “Tutorial: Análisis De Sentimiento: Clasificación Binaria - ML.NET.” *Tutorial: Análisis De Sentimiento: Clasificación Binaria - ML.NET | Microsoft Docs*, docs.microsoft.com/es-es/dotnet/machine-learning/tutorials/sentiment-analysis-model-builder.

[20] *Stanford CoreNLP*, stanfordnlp.github.io/CoreNLP/.

[21] “Swiss Center For Affective Sciences Swiss Center For Affective Sciences.” *UNIGE*, 20 Feb. 2019, www.unige.ch/cisa/research/materials-and-online-research/research-material/.

[22] tpsatish95. “tpsatish95/Emotion-Detection-from-Text.” *GitHub*, github.com/tpsatish95/emotion-detection-from-text/tree/master/datasets.

[23] Azmin, Sara, and Kingshuk Dhar. “Emotion Detection from Bangla Text Corpus Using Naïve Bayes Classifier.” *2019 4th International Conference on Electrical Information and Communication Technology (EICT)*, 2019,

doi:10.1109/eict48899.2019.9068797.

[24] “Keyword Based Emotion Word Ontology Approach for Detecting Emotion Class from Text.” *International Journal of Science and Research (IJSR)*, vol. 5, no. 5, 2016, pp. 1636–1639., doi:10.21275/v5i5.nov163818

[25] Rabeya, Tapasy, et al. “A Survey on Emotion Detection: A Lexicon Based Backtracking Approach for Detecting Emotion from Bengali Text.” *2017 20th International Conference of Computer and Information Technology (ICCIT)*, 2017, doi:10.1109/iccitechn.2017.8281855.

[26] Shivhare, Shiv Naresh. “Emotion Detection from Text.” *Computer Science & Information Technology (CS & IT)*, 2012, doi:10.5121/csit.2012.2237.