UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

CARACTERIZACIÓN DE LOS ESTILOS DE CONDUCCIÓN MEDIANTE SMARTPHONES, DISPOSITIVOS OBD-II Y REDES NEURONALES

Realizado por:
Javier E. Meseguer Anastasio

Directores:
Dr. Carlos Miguel Tavares Calafate
Dr. Juan Carlos Cano Escribá

Septiembre 2012

Índice.

Índice de Figuras.

ÍNDICE DE GRÁFICAS.

ÍNDICE DE TABLAS.

1 INTRODUCCIÓN.

1.1 Motivación.

Los dispositivos móviles han experimentado un gran avance tecnológico en estos últimos años, y se ha pasado de utilizar microprocesadores multi-núcleo en máquinas de alto rendimiento, a llevar en el bolsillo móviles que en su interior incorporan este tipo de microprocesadores. Por otra parte el estándar *On Board Diagnosis* (OBD-II), disponible desde hace varios años, permite acceder de forma sencilla al *Electronic Control Unit* (ECU) mediante un conector Bluetooth OBD-II. Este interfaz de conexión permite la conectividad entre el dispositivo móvil y el vehículo, y se puede adquirir por poco mas de 20€. En el mercado ya existen aplicaciones comerciales como Torque [14] o CarGaugeLite [15] que muestran la potencialidad de esta simbiosis.

El espectro de posibilidades que surgen al combinar el automóvil y el smartphone es amplísimo, como por ejemplo realizar el diagnóstico del coche a través del móvil asumiendo las tareas que hace la unidad On Board Unit (OBU) del coche, o bien enviar los datos recogidos a una plataforma donde se pueda realizar el diagnóstico o mantenimiento del sistema, detectando posibles fallos, etc.

La propuesta que se pretende llevar a cabo con este trabajo fin de Máster no consiste únicamente en la captura de datos y su presentación al usuario mediante el uso de gráficas, mapas, estadísticas, etc., sino que pretende ir mucho más allá, aplicando técnicas de minería de datos para analizar y generar una clasificación sobre los estilos de conducción en base a un análisis de las características de la vía sobre la que ha realizado la ruta. El objetivo final persigue ayudar al conductor a corregir malos hábitos en su forma de conducción, ofreciendo consejos útiles para conseguir mas ahorro de combustible. Es bien conocido que una conducción inteligente puede llevarnos a un menor gasto de gasolina, con el consiguiente impacto sobre el medio ambiente.

1.2 Objetivos.

Los objetivos de esta tesina de Máster son el diseño e implementación de una plataforma en la que se pueda realizar, mediante minería de datos, estudios sobre los estilos de conducción de los usuarios. A partir de la velocidad, la aceleración y las revoluciones del motor obtenidos se implementará, mediante un algoritmo de entrenamiento basado en redes neuronales capaz de caracterizar el tipo de vía por la que se circula en cada tramo, el estilo de conducción de cada usuario. En un futuro se podría recopilar esta información para poder utilizarla en aplicaciones orientadas a mejorar la seguridad vial.

El objetivo general se puede detallar en 3 bloques claramente definidos:

1. Realización de una aplicación para smartphones. Nos hemos decantado por realizarla en Android [3] para así fomentar el software libre, pero de igual manera se podría haber realizado en iOS [4] ya que entre ambos copan el 90% del mercado [4].
   Mediante el uso de un interface Bluetooth OBD-II (ELM327) la aplicación recopila información como la velocidad, la aceleración, las revoluciones del motor, la posición del pedal del acelerador y la posición geográfica del vehículo mediante GPS.
   Una vez recopilada la información el usuario enviará los datos de la ruta realizada al portal web donde éstos serán analizados.

2. Realización de un portal Web que recopile todos los datos enviados por los usuarios. Hemos abogado en este caso por realizarla con herramientas de software libre como Apache, PHP y Joomla.
   Después de recopilar los datos de los usuarios utilizaremos aquellos más representativos para entrenar las Redes Neuronales que deseamos obtener y que constituye el tercer objetivo de este trabajo.

3. Análisis de los datos almacenados en el servidor mediante Redes Neuronales. Los datos obtenidos, serán utilizados para poder entrenar una Red Neuronal mediante el algoritmo de *backpropagation* que proporciona buenos resultados en problemas de tipo clasificadores, como es el caso del proyecto. Una vez entrenada con los parámetros obtenidos se ha integrado, junto al portal web, con el objetivo de que cada usuario pueda averiguar cual es su perfil como conductor.

2 ESTADO DEL ARTE.

2.1 Trabajos previos.

A medida que avanza la tecnología en los teléfonos móviles, estos dispositivos logran alcanzar mayores potencias de cómputo. En el mundo de la investigación, esta potencia hace que se abran nuevas líneas de investigación y que, además, muchas de ellas sean económicamente provechosas, como por ejemplo el prototipo de una unidad a bordo desarrollado por Hernandez et al. [12] que permitía al conductor comunicarse con el vehículo al igual que con otros dispositivos disponibles (PDAs, móviles, redes de sensores y otros) y con una infraestructura de carretera, con el fin de obtener servicios de transporte inteligente. En ese trabajo se propusieron dos nuevos servicios basados en comunicaciones vehículo-a-infraestructura: a) Informes de tráfico en tiempo real y b) soporte a la eco-conducción.

Respecto a otras líneas de trabajo más relacionadas con la propuesta realizada en este trabajo de Máster, cabe destacar el trabajo de Chen et al. [13], que propusieron una plataforma vehicular Android/OSGi que permite diagnosticar y gestionar el estado del sistema de una plataforma vehicular de forma remota, utilizando la inteligencia visual para actualizar continuamente sus servicios de aplicación basados en el contexto sin la intervención del usuario. Los experimentos realizados en un banco de pruebas vehicular mostraron que la plataforma Android/OSGi permite desarrollar aplicaciones más ligeras y con un mayor rendimiento que una plataforma Android pura, especialmente cuando hay que realizar operaciones complicadas.

Parte I – Aplicación Móvil y Web.

3 TECNOLOGIAS UTILIZADAS.

3.1 Dispositivo Móvil – Android.

Android es un sistema operativo inicialmente pensado para teléfonos móviles, al igual que iOS, Symbian y Blackberry OS. Lo que lo hace diferente es que está basado en Linux, un núcleo de sistema operativo **libre, gratuito y multiplataforma** y por ello lo hemos utilizado en este trabajo (versión 2.2 –Froyo- basado en el Kernel de Linux2.6.32 [5][6]).

Android era un sistema operativo para móviles prácticamente desconocido hasta que en 2005 Google lo compró. Hasta noviembre de 2007 sólo hubo rumores, pero en esa fecha se lanzó la **Open Handset Alliance**, que agrupaba a muchos fabricantes de teléfonos móviles, chipsets y Google, y se proporcionó la primera versión de Android, junto con el SDK para que los programadores empezaran a crear sus aplicaciones para este sistema.

El sistema permite programar aplicaciones para una variante de Java llamada Dalvik [6], y proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java. Android permite controlar dispositivos mediante bibliotecas desarrolladas o adaptadas por Google mediante el lenguaje de programación Java.

3.1.1 Características de Android.

La gran ventaja que tiene Android es que se adapta muy bien al IDE de desarrollo de la herramienta Eclipse, ofreciendo un *plugin* que permite programar y analizar el código de manera que resulte cómodo en el entorno de desarrollo que se dispone, el cual incluye un emulador del dispositivo, herramientas para la depuración, memoria y perfiles de rendimiento.

Las características principales de Android son:

- **Máquina virtual Dalvik:** Se dispone de todas las configuraciones posibles de configuración de móviles, lo que permite probar soluciones sin tener necesidad de disponer de una gran cantidad de dispositivos móviles, estando optimizada para estos dispositivos.
- **Navegador integrado:** Basado en el motor Open Source Webkit, la aplicación lo utiliza para comunicarse con el servidor y realizar diversas operaciones, como darse de alta como usuario en el portal *http://www.drivingstyles.info*, ver las rutas que se tienen almacenadas en el servidor y proporcionar información del comportamiento de conducción.
- **Framework de aplicaciones:** Permite reemplazar y reutilizar los componentes.

- **Gráficos optimizados:** Tiene implementada una biblioteca de gráficos 2D. Los gráficos 3D están basados en la especificación OpenGL ES 1.0 (la aceleración hardware es opcional). La aplicación Android que hemos desarrollado utiliza las librerías 2D para la generación de las graficas de velocidad, aceleraciones y revoluciones del motor, tanto las generadas en el momento de la captura de datos, como las almacenadas previamente en la propia base de datos.
- **SQlite:** Base de datos para almacenamiento estructurado que se integra directamente con las aplicaciones.
- **Multimedia:** Soporte para medios de audio, video e imágenes (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- **Telefonía GSM.**
- **Cámara.**
- **Bluetooth:** En este proyecto se ha implementado una clase *com.driving.styles.obdcodes basada en un principio en un proyecto de software libre, mantenido en parte por Google (*http://code.google.com/p/elm327/*).* Que realiza la comunicación entre el smartphone y el OBD-II del vehículo.
- **EDGE, 3G y WiFi**. Las librerías típicas que utiliza Android para la conexión y envío de paquetes con los servidores.
- **GPS y Brújula:** Solo se ha utilizado el GPS, para posicionar el vehículo sobre la vía en cuestión y poder comprobar el correcto funcionamiento de la Red Neuronal.
- **Acelerómetro.** En el proyecto se ha utilizado la clase `android.hardware.SensorManager` para obtener las aceleraciones que se producen sobre el móvil.

3.1.2 Arquitectura de Android.

La arquitectura interna de la plataforma Android se puede dividir en cinco grandes componentes: Kernel de Linux, bibliotecas, entorno de ejecución, marco de aplicación y aplicaciones.

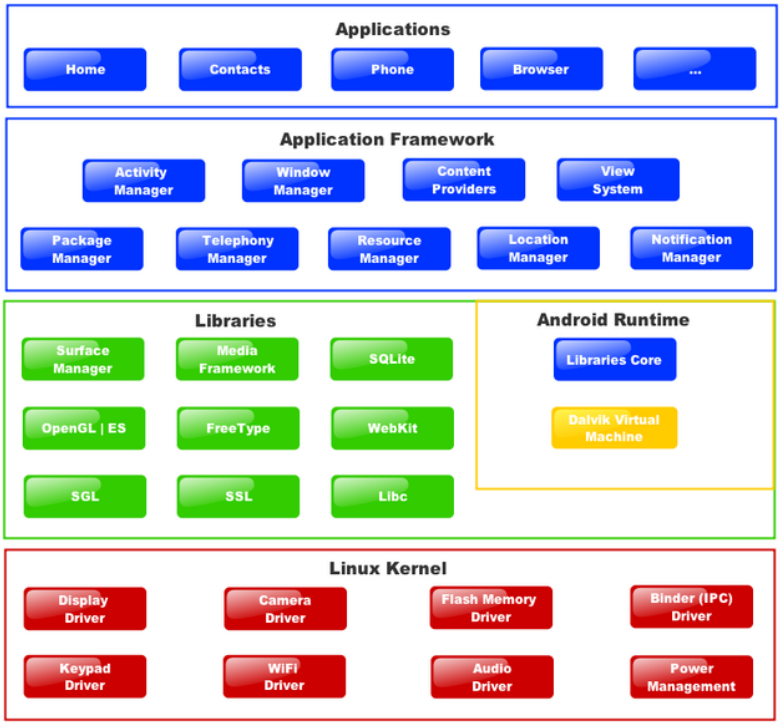


Figura 3.1.2.1: Arquitectura de Android.

3.1.2.1 Kernel de Linux.

El núcleo del sistema operativo Android es un kernel Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, el cual ha sido adaptado a las características del hardware en el que se ejecutará Android (normalmente un smartphone).

Proporciona una capa de abstracción para los elementos hardware a los que tienen que acceder las aplicaciones. Esto permite que se pueda acceder a esos componentes sin necesidad de conocer el modelo o características precisas de los que están instalados en cada teléfono. De esta forma, si una aplicación necesita, por ejemplo, la brújula, podrá utilizar la que incluya el teléfono. Para cada elemento hardware del teléfono existe un controlador (*driver*) dentro del kernel que permite utilizarlo desde el software.

Además de proporcionar controladores hardware, el kernel se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación (*networking*), etc.

3.1.2.2 Bibliotecas.

La capa que se sitúa justo sobre el kernel la componen las bibliotecas nativas de Android. Estas bibliotecas están escritas en C o C++ y están compiladas para la arquitectura hardware específica del teléfono, tarea que normalmente realiza el fabricante, que también se encarga de instalarlas en el terminal antes de ponerlo a la venta. Su cometido es proporcionar funcionalidad a las aplicaciones, para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma más eficiente.

3.1.2.3 Entorno de ejecución.

El entorno de ejecución de Android, aunque se apoya en las bibliotecas enumeradas anteriormente, no se considera una capa en sí mismo, dado que también está formado por bibliotecas. En concreto, las bibliotecas esenciales de Android, que incluyen la mayoría de la funcionalidad de las bibliotecas habituales de Java así como otras específicas de Android.

3.1.2.4 Marco de aplicación.

La siguiente capa la forman todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones y que, obviamente, se apoyan en las bibliotecas y en el entorno de ejecución que ya hemos detallado. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos a través de la máquina virtual Dalvik [6].

3.1.2.5 Aplicaciones.

La capa superior de esta pila software la forman, como no podría ser de otra forma, las aplicaciones. Se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, tanto las nativas (programadas en C o C++) como las administradas (programadas en Java), y tanto las que vienen de serie con el dispositivo como las instaladas por el usuario, como seria el caso de la aplicación de la presente tesina.

3.1.3 Bluetooth Android.

El SDK de *Android* incluye soporte para la pila de interconexión Bluetooth. El API Bluetooth permite la conexión inalámbrica con otros dispositivos Bluetooth, en nuestro caso el OBD-II, habilitando características punto-a-punto y multipunto.

La aplicación realizada en este trabajo de fin de Máster realiza las siguientes tareas con respecto a la conexión bluetooth:

1. Escaneo de otros dispositivos Bluetooth.
2. Consulta del adaptador Bluetooth para identificar los dispositivos Bluetooth emparejados.
3. Establecer canales RFCOMM.
4. Conectarse con otros dispositivos a través del service discovery (SDP).
5. Transferir/recibir datos con otros dispositivos.

Las clases que podemos encontrar en la API de Android son:

- **BluetoothAdapter:** Representa el adaptador Bluetooth local y es el punto de entrada para todas las interacciones Bluetooth. Mediante su uso se pueden descubrir otros dispositivos Bluetooth, una lista de los dispositivos emparejados, instanciar un "BluetoothDevice" usando una dirección MAC conocida y crear un "BluetoothServerSocket" para escuchar las comunicaciones de otros dispositivos.
- **BluetoothDevice:** Representa un dispositivo Bluetooth remoto. Se usa para solicitar una conexión con un dispositivo remoto a través de un "BluetoothSocket" o consultar información acerca del dispositivo (nombre, dirección, clase, estado, etc.)
- **BluetoothSocket:** Representa el interface para un socket Bluetooth (parecido a un socket TCP). Es el punto de conexión que permite a una aplicación el intercambio de datos con otro dispositivo Bluetooth mediante el uso de InputStream y OutputStream.
- **BluetoothServerSocket:** Representa un server socket abierto que mantiene la escucha para peticiones de entrada (parecido a un ServerSocket TCP). Para la conexión de dos dispositivos Android, un dispositivo debe abrir el server socket con esta clase. Cuando el dispositivo Bluetooth remoto hace la petición de conexión a ese dispositivo, el "BluetoothServerSocket" retornará un "BluetoothSocket" cuando la conexión sea aceptada.
- **BluetoothClass:** Describe las características generales y capacidades del dispositivo Bluetooth. Es un conjunto de propiedades de solo lectura. Sin embargo, no es fiable del todo ya que no describe todos los perfiles Bluetooth y servicios soportados por el dispositivo.

3.1.4 Geolocalización y Mapas.

El SDK de *Android* incluye dos *paquetes* que proporcionan soporte para montar servicios de localización: *android.location* y *com.google.android.maps.* El primero contiene varias clases relacionadas con los servicios de localización, e incluye el servicio *LocationManager,* el cual proporciona una *API* para determinar la localización del dispositivo móvil. *LocationManager* no puede ser instanciado directamente, sino que se debe obtener un controlador, tal y como muestra la figura 3.1.4.1.

```
myLocationManager= (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

Figura 3.1.4.1: Creación de un LocationManager.

Cuando se obtiene el control de un LocationManager, se pueden realizar las siguientes acciones, como se observa en la figura 3.1.4.2.

- Solicitar una lista de todos o algunos de los LocationProviders disponibles.
- Registrarse o dejar de estar registrado para actualizaciones periódicas de la posición.
- Registrarse o dejar de estar registrado de eventos cuando el dispositivo se encuentra en determinada ubicación (por ejemplo, cerca de otros dispositivos).

```
//Solicitación de un LocationProvider concreto (gps)
myLocation = myLocationManager.getLastKnownLocation("gps");
//Registro para actualizaciones periódicas (2 arg) y para eventos (3 arg)
myLocationManager.requestLocationUpdates(strProvider, MINIMUM_TIME_BETWEEN_UPDATE,
MINIMUM_DISTANCECHANGE_FOR_UPDATE, listener );
```

Figura 3.1.4.2: Acciones posibles con un *LocationManager.*

En el segundo *paquete* (*com.google.android.maps*), se obtienen varias clases relacionadas con el *renderizado*, control y dibujo sobre los mapas de la ruta. La más importante es la clase *MapView* que automáticamente dibuja un mapa básico de *Google Maps* cuando se añade al *layout* de la aplicación, figura 3.1.4.3.

```
// Crea un Mapa y lo muestra por pantalla
myMapView = new MapView(this, "");
setContentView(myMapView);
```

Figura 3.1.4.3: Inclusión de mapas en el módulo mapa.

Para poder obtener un *MapView*, se necesita previamente un *API Key* de *Google Maps* (ver ANEXO 10.1). Además, para poder hacer uso de esta clase, se ha de añadir el permiso en el manifiesto dado que no es un *paquete* estándar de *Android.* Como último requisito, es necesario tener la configuración regional del ordenador donde corre el emulador seleccionada en Estados Unidos para que reconozca la puntuación de las coordenadas.

3.1.5 Acelerómetro Android.

En Android el acceso a los servicios de sensores hardware se realizan a través de las clases del paquete “android.hardware”.

Las clases e interfaces más importantes de éste paquete son:

- **SensorManager:** Es la clase que permite acceder a los sensores del dispositivo.
- **Sensor:** Esta clase representa un sensor. La clase “Sensor” acepta actualmente 12 tipos de sensores, tales como acelerómetro, giroscopio, orientación, sensor de luz, etc.
- **SensorEvent:** Esta clase representa un evento de un sensor y recoge información como puede ser el tipo del sensor, el *timestamp*, la precisión y los datos del sensor.
- **SensorEventListener:** Se trata de una interfaz que se usa para recibir notificaciones desde el “SensorManager” cuando los valores del sensor han cambiado.

En la aplicación DrivingStyles desarrollada se ha utilizado el sensor acelerómetro. Este sensor recoge todos sus valores en la unidad de aceleración m/s^2.

El valor devuelto por el sensor se trata de un vector de longitud 3, donde:

- **Valor[0]:** Aceleración menos Gx en el eje X.
- **Valor[1]:** Aceleración menos Gy en el eje Y.
- **Valor[2]:** Aceleración menos Gz en el eje Z.

```
// Escuchamos desde el listener del acelererómetro
private static SensorEventListener sensorEventListener =
            new SensorEventListener() {

            private float x = 0;
            private float y = 0;
            private float z = 0;

            public void onAccuracyChanged(Sensor sensor, int accuracy) {}

            public void onSensorChanged(SensorEvent event) {
                x = event.values[0];
                y = event.values[1];
                z = event.values[2];

                listener.onAccelerationChanged(x, y, z);
            }

};
```

Figura 3.1.5.1: Captura de valores del acelerómetro en la aplicación Android.

3.2 OBD-II.

OBD-II es la abreviatura de On Board Diagnostics II (Diagnóstico de Abordo II), es la segunda generación de la electrónica obligatoria que deben disponer todos los coches de los Estados Unidos (la primera generación fue el OBD-I, y un endurecimiento en los límites de emisiones en 1996 llevó a la creación del OBD-II.). Este sistema incorpora dos sensores de oxígeno (sonda Lambda), uno ubicado antes del catalizador y otro después del mismo, pudiendo así comprobarse el correcto funcionamiento de éste.

Si se detecta un problema o un fallo, el sistema OBD-II enciende una lámpara de advertencia en el salpicadero llamada MIL (*Malfunktion Indicator Lamp*) que avisa al conductor de un fallo. La lámpara de advertencia normalmente lleva la inscripción "*Check Engine*" o "*Service Engine Soon*". El sistema también guarda información importante sobre los fallos detectados para que un mecánico pueda encontrar y resolver el problema.

En los Estados Unidos todos los vehículos de gasolina desde 1996 deben contar con sistemas de OBD-II, al igual que todos los vehículos de pasajeros y camiones diésel a partir de 1997. En Europa, según la Directiva 98/69EG, los coches a gasolina desde el año 2000 en adelante, los coches a diesel a partir del 2003 y los camiones desde el 2005 tienen que estar provistos del OBD.

OBD-I: Fue diseñado para detectar fallos eléctricos en el sistema y en los componentes. La luz del denominada MIL (Malfunktion Indicator Lamp) se apaga si el problema de emisiones se corrige por si solo.

OBD-II: Monitorea el comportamiento de los sistemas de emisión y de los componentes, así como los fallos eléctricos y almacena información para un uso posterior.
La lámpara MIL se mantiene encendida hasta que hayan pasado 3 ciclos de conducción consecutivos, sin que el problema reincida.
La memoria se borra después de 40 arranques en frío. Si se trata del escaneo del combustible se necesitan 80 arranques en frío.

OBD-I: ESCANEADOS REQUERIDOS (California 1988, Federal 1994)

- Sensor de oxigeno.
- Sistema EGR.
- Sistema de reparto de combustible.
- PCM.

OBD-II: ESCANEADOS REQUERIDOS (Federal 1996)

- Eficiencia del catalizador.
- Fuego (Misfire).
- Control de combustible.
- Respuesta del sensor de oxigeno.
- Calefactor del sensor de oxigeno.
- Detallado de componentes.
- Emisiones expulsadas.
- Sistema de aire secundario (si esta equipado).
- EGR.

El estándar de OBD-II implementa varios modos de trabajo; esto significa que, dependiendo la información a la que se desee acceder, se necesita un modo diferente. Una vez dentro de ese modo de trabajo, se ofrece un extenso número de parámetros para acceder a dicha información. Se incluyen los siguientes modos (Tabla 3.2.1):

MODO	CARACTERÍSTICAS
Modo 01	Identificación de Parámetro (PID). Es el acceso a datos en vivo de valores de salidas y entradas a la ECU (*Engine Control Unit*).
Modo 02	Acceso a Cuadro de Datos Congelados. La ECU toma una muestra de todos los valores relacionados con las emisiones en el momento exacto de ocurrir un fallo.
Modo 03	Permite extraer de la memoria de la ECU todos los códigos de fallo (DTCs) almacenados.
Modo 04	Se pueden borrar todos los códigos almacenados en la PCM (*Power Train Control Module*) incluyendo los DTCs y el cuadro de datos grabado.
Modo 05	Devuelve los resultados de las pruebas realizadas a los sensores de oxigeno para determinar el funcionamiento de los mismos y la eficiencia del convertidor catalítico.
Modo 06	Permite obtener los resultados de todas las pruebas de abordo.
Modo 07	Permite leer de la memoria de la ECU todos los DTCs pendientes.

Tabla 3.2.1: Modos de trabajo del estándar OBD-II.

Aunque estos modos son los que contempla el estándar OBD-II, los fabricantes de automóviles han ido ampliado sus funciones para poder controlar y gestionar muchos más aspectos del vehículo, como por ejemplo leer cualquier código de error registrado por la centralita, activar o desactivar funciones del vehículo, ABS, inyección, etc.

3.2.1 Hardware y protocolos del OBD-II.

El OBD-II tiene un interfaz hardware estándar de 16 pines (conector J1962 hembra), ver figura 3.2.1.1. El conector cumple las especificaciones según la normativa ISO 15031-3:2004, que específica incluso donde debe estar situado el conector, aunque realmente se suele encontrar en cualquier posición, siempre debajo del salpicadero.

Pin 2	J1850 Bus+
Pin 4	Chassis Ground
Pin 5	Signal Ground
Pin 6	CAN High (J-2284)
Pin 7	ISO 9141-2 K Line
Pin 10	J1850 Bus
Pin 14	CAN Low (J-2284)
Pin 15	ISO 9141-2 L Line
Pin 16	Battery Power

Tabla 3.2.1.1:Pines del interfaz OBD-II.

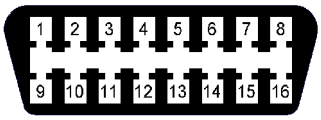


Figura 3.2.1.1: Pines del interfaz OBD-II.

El OBD-II ofrece cinco protocolos diferentes, aunque la mayor parte de vehículos implementan solo uno de esos protocolos. Todos los pines de conexión de OBD-II usan el mismo conector, pero según el protocolo difieren en qué pines, excepto el pin 4 (Negativo batería) y el pin 16 (Positivo batería).

Los protocolos son los siguientes:

SAE J1850 PWM	Modulación Ancho de Pulso (P*ulse Width Modulation*) utilizado por Ford USA.
SAE J1850 VPW	Ancho de Pulso Variable (*Variable Pulse Width*) lo utiliza GM USA
ISO 9141-2	En vehículos Europeos, Asiáticos y Chrysler con variantes.
ISO 14230 KWP2000	(Keyword Protocol 2000) utilizado por el grupo VAG.
ISO 15765 CAN	

Tabla 3.2.1.2: Protocolos del estándar OBD-II.

En general, los productos europeos, muchos asiáticos y el fabricante Chrysler aplican el protocolo ISO 9141. General Motors utiliza el SAE J1850 VPW y Ford aplica patrones de comunicación SAE J1850 PWM.

3.2.2 ELM327 Bluetooth.

Los datos que transfiere el vehículo al OBD-II sigue varios protocolos, pero ninguno de ellos se puede usar directamente en nuestro un dispositivo móvil. Necesitamos un dispositivo como el ELM327 [8] que está diseñado para actuar como interfaz entre los puertos OBD y el interfaz estándar RS232.

Una vez establecida la conexión con el ELM327 mediante alguno de los protocolos vistos anteriormente, el dispositivo mandará un mensaje con la versión del dispositivo ELM327 y posteriormente el carácter “>”, lo que significa que el dispositivo está listo y preparado para recibir caracteres a través del puerto RS232.

Los caracteres enviados al dispositivo ELM327 [8] pueden ser interpretados por éste, determinando si se trata un comando de configuración del propio dispositivo, o un mensaje para el vehículo; en ese caso el mensaje se reformatearía y se envía al interfaz OBD. El dispositivo puede determinar a quién va dirigido el mensaje ya que los mensajes que empiezan mediante “AT” son para la configuración del dispositivo ELM327, mientras que los comandos OBD para el vehículo solo tienen permitido contener códigos ASCII específicos.

Para cualquiera de los dos tipos de comando, el mensaje debe terminar con el carácter de retorno de carro (Hex ‘0D’) para que pueda ser evaluado. En caso contrario, se activa un temporizador automáticamente que, tras 20 segundos, abortará el mensaje.

Figura 3.2.2.1: Dos de los modelos OBD-II bluetooth utilizados en el proyecto.

3.3 PORTAL WEB: Apache-MySQL-Php-Joomla.

3.3.1 Apache.

El servidor web utilizado en el proyecto ha sido el servidor Apache de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows y Macintosh (desarrollado dentro del proyecto HTTP Server de la Apache Software Foundation.). El hecho de que sea de código abierto ha sido fundamental para su elección, así como que es multiplataforma. El servidor consta de una sección llamada núcleo y diversos módulos que aportan mucha de la funcionalidad que podría considerarse básica para un servidor web. Algunos de los módulos más importantes son:

1. *mod_ssl* - Comunicaciones Seguras vía TLS.
2. *mod_rewrite* - Reescritura de direcciones.
3. *mod_php* - Páginas dinámicas en Php.

3.3.2 MySQL.

Como sistema de gestión de la base de datos se ha utilizado MySQL. Es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones, este gestor de bases de datos es, probablemente, el gestor más usado en el mundo del software libre, debido a su gran velocidad y facilidad de uso. La gran aceptación que tiene es debida, en parte, a que existen infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración.

3.3.3 Php.

Php es un acrónimo recursivo que significa “Php Hypertext Preprocessor”. Es un lenguaje de programación usado en el gestor de contenidos como por ejemplo Joomla, lenguaje de programación interpretado y diseñado originalmente para la creación de páginas web dinámicas. Es utilizado principalmente en interpretación del lado del servidor, pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas, incluyendo aplicaciones con interfaz gráfica utilizando las bibliotecas Qtro y GTK+.

3.3.4 SQL.

El Lenguaje de Consulta Estructurado SQL (Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional, permitiendo ejecutar consultas con el fin de recuperar, de una forma sencilla, información de interés de una base de datos, así como también hacer cambios sobre la misma.

3.3.5 Otras tecnologías utilizadas Ajax, CSS y JavaScript.

En el proyecto de la tesina del Máster también se han utilizado otras tecnologías que se describen a continuación.

3.3.5.1 Ajax.

AJAX (Asynchronous JavaScript And XML) es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, la velocidad y la usabilidad en las aplicaciones.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como Javascript y Document Object Model (DOM).

3.3.5.2 Javascript.

JavaScript es un lenguaje interpretado, es decir, que no requiere compilación, con una sintaxis semejante a la del lenguaje Java y el lenguaje C, que permite crear aplicaciones específicamente orientadas a su funcionamiento en webs. Usando JavaScript, se pueden crear páginas HTML dinámicas que procesen la entrada del usuario y que sean capaces de gestionar datos persistentes usando objetos especiales, archivos y bases de datos relacionales.

3.3.5.3 CSS.

Las hojas de estilo en cascada (Cascading Style Sheets, CSS) son un lenguaje formal utilizado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML), separando la estructura de un documento de su presentación. El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

CSS permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas Web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS en las que aparezca ese elemento.

4 VISIÓN GENERAL DE LA ARQUITECTURA DRIVINGSTYLES.

En la figura 4.1 podemos ver de un modo muy gráfico el ciclo de vida de la arquitectura DrivingStyles desde el punto de vista del usuario. Lo primero que necesita realizar el usuario es entrar en el portal http://www.drivingstyles.info y descargarse la aplicación, de forma gratuita, para Android. A continuación deberá darse de alta en ella para poder enviar sus datos; en el momento esté registrado podrá también acceder a la zona de usuario, y visualizar sus estadísticas y rutas.

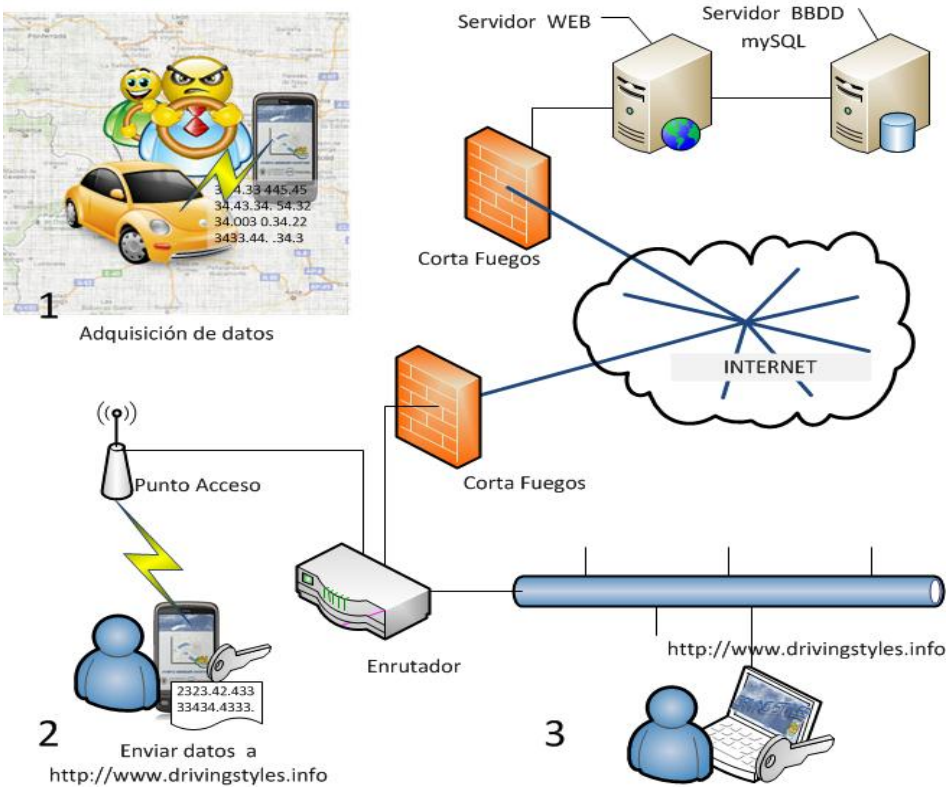


Figura 4.1: Visión general de la arquitectura DrivingStyles.

Una vez instalada la aplicación Android en el dispositivo móvil, y conectado el ELM327 bluetooth en el automóvil, solamente nos queda emparejar ambos dispositivos para comenzar la adquisición de datos (1). Una vez terminado el recorrido y almacenadas las rutas, éstas son enviadas al servidor, el cual comprueba el usuario y la contraseña para aceptar el fichero XML recibido (2), almacenándose en el. El usuario podrá consultar todas las rutas que ha enviado al servidor accediendo a las gráficas y mapas del itinerario (3).

De un modo más conceptual, podemos ver el esquema de la figura 4.2, donde los 3 grandes bloques del presente trabajo que se han presentado anteriormente quedan más claros.

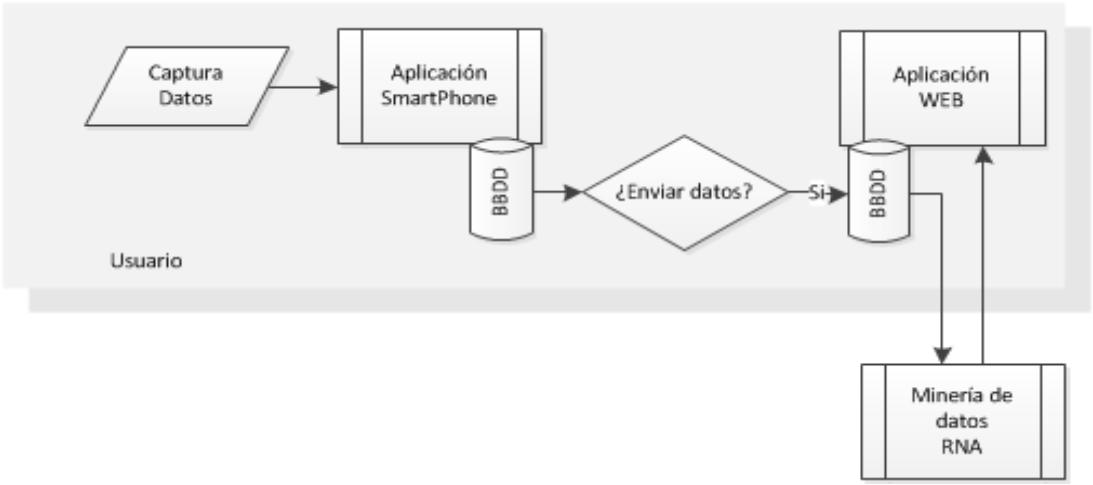


Figura 4.2: Diagrama de bloques de la arquitectura DrivingStyles.

También se puede observar en el diagrama anterior (figura 4.2) que las bases de datos de ambas aplicaciones, smartphone y web, están relacionadas y son casi idénticas, salvo alguna pequeña diferencia, como por ejemplo que en la aplicación web realizamos la parte relacionada con las redes neuronales ya que necesita de tablas adicionales.

Dos de las tablas de la aplicación web se han creado para utilizarlas en la generación de los ficheros que se usan en el entrenamiento y validación de la red neuronal, las muestras de rutas que se consideran representativas y que podemos clasificar, las almacenamos en la base de datos clasificándolas según el tipo de ruta y el estilo de conducción para su posterior generación del fichero.

5 APLICACIÓN ANDROID: DRIVINGSTYLES.

La aplicación esta dividida en varios módulos, típicamente como cualquier aplicación para dispositivos móviles (cada una de las pantallas corresponde con cada uno de los módulos de la aplicación), lo que permite distinguir cada una de las funcionalidades de ésta, además de facilitar al usuario disponer de una interfaz que le sea familiar, parecida a otras aplicaciones ya utilizadas.

Empezaremos la exposición de la aplicación como si de un usuario nuevo se tratase.

La aplicación para Android se puede descargar de forma gratuita desde el propio portal web DrivingStyles en la URL: http://www.drivingstyles.info/index.php/en/ o desde el código QR de la figura 5.1.

Figura 5.1: Código QR para la descarga de la aplicación Android.

Una vez instalada, al iniciar la aplicación aparece una pantalla con una presentación del grupo de investigación y la universidad, además de realizar precargas de datos requeridos para que la aplicación funcione adecuadamente. Por ejemplo, si es la primera vez que se ejecuta la aplicación, se crea la base de datos requerida.

Figura 5.3: Pantalla de inicio.

Figura 5.2: Aplicación funcionando en un vehículo.

5.1 Opciones de configuración.

Observando el interfaz del módulo de configuración se pueden observar las diferentes secciones de las que se compone la aplicación, donde cada una de las secciones está destinada a la configuración de un modulo de la aplicación.

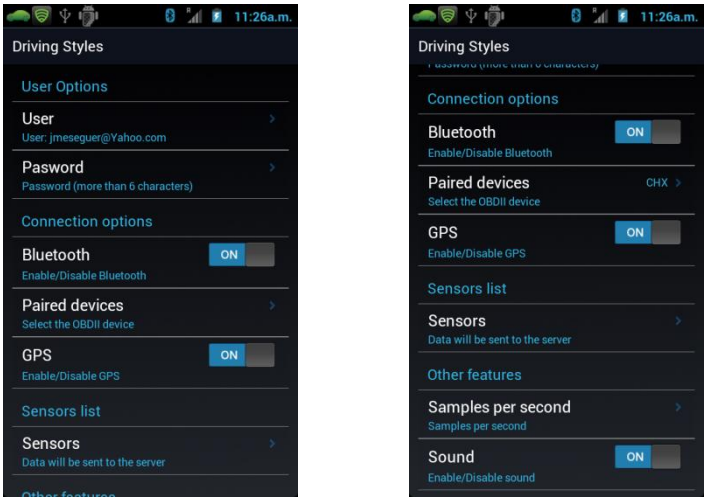


Figura 5.1.1: Opciones de configuración de la aplicación.

El módulo lo podemos subdividir en:

Creación del usuario: Se necesita crear un usuario con un password que debe coincidir con el que tengamos creado en el portal web, ya que al enviar los datos de la captura, el servidor los guarda en la base de datos y en el directorio de éste, validando el password que se envía con el fichero de datos.

Opciones de conexión: Antes de empezar a tomar muestras se debe emparejar el teléfono con el OBD-II del coche, como cualquier otro dispositivo bluetooth. Para realizar la conexión entre el dispositivo móvil y el interfaz Bluetooth ELM327, es necesario tener emparejado en el dispositivo el interfaz ELM237, de eso se encarga la opción “Paired Devices”. En ella nos encontramos todos los dispositivos emparejados hasta el momento. Una vez lo seleccionemos entre la lista de dispositivos la aplicación podrá acceder a su dirección MAC para posteriormente realizar la conexión entre interfaz ELM327 y dispositivo móvil.

Activación del GPS: No es obligatorio tenerla activada para la captura de datos pero si que es necesaria para el estudio que se esta realizando, las variables obtenidas del motor del automóvil se graban junto al posicionamiento del automóvil.

Selección de sensores: Nos permite seleccionar los sensores a mostrar en la aplicación y que capturamos. Actualmente podemos disponer del número de revoluciones del motor (rpm), la posición del pedal de aceleración y la velocidad instantánea del automóvil. Las aceleraciones del coche, como posteriormente veremos, se calcula

mediante el sensor de la velocidad. La selección de los sensores es dependiente de bluetooth, de forma que si éste no esta activado no se podrán seleccionar.

Selección de muestras por segundo: Número de capturas de datos por segundo (por defecto 2 por segundo.), no es recomendable cambiarlo porque podemos capturar datos erróneos o perder muestras.

Este módulo es importante para que el funcionamiento de la aplicación sea el correcto, por lo que la primera vez que se inicie la aplicación se indica que se debe configurar. Si la aplicación está mal configurada lo indicara con mensajes de error.

5.2 Módulo principal.

El módulo principal de la aplicación, es el encargado de lanzar los procesos background de captura de datos enviados por el OBD-II, el GPS y el acelerómetro del móvil (figura 5.2.1).

Figura 5.2.1: Pantalla principal.

Además de mostrar los sensores que estamos monitorizando, podemos realizar varias acciones posibles en paralelo sin que afecte a la captura de los datos.

Opciones posibles:

- Iniciar y parar las capturas de datos del *Electronic Control Unit* (ECU), del acelerómetro del móvil y del GPS, mediante los botones.
- Visualizar listado de rutas realizadas.
- Visualizar en el mapa la posición actual del vehículo, si no se dispone conexión a internet solo podrá visualizar el trazado de la ruta, con conexión (Wifi, 3G, etc.) podremos observar la ruta realiza sobre el mapa.
- Visualizar las graficas de velocidad, revoluciones y aceleración del vehículo en una ventana de 10 segundos.

Se ha configurado la aplicación para capturar 2 muestras por segundo de cada uno de los datos del vehículo, esta frecuencia nos permite trabajar con una sensibilidad adecuada sin llegar a saturar el canal de comunicaciones serie que tiene un ancho de banda muy limitado.

5.2.1 Diagrama de Flujo.

El diagrama que se muestra a continuación muestra el comportamiento de la aplicación en el momento de la ejecución de la actividad principal, es decir, qué comprobaciones realiza para que posteriormente la ejecución normal de la aplicación no se vea afectada por parámetros que deberían estar establecidos, y las acciones relacionadas con las diferentes funcionalidades de la misma.

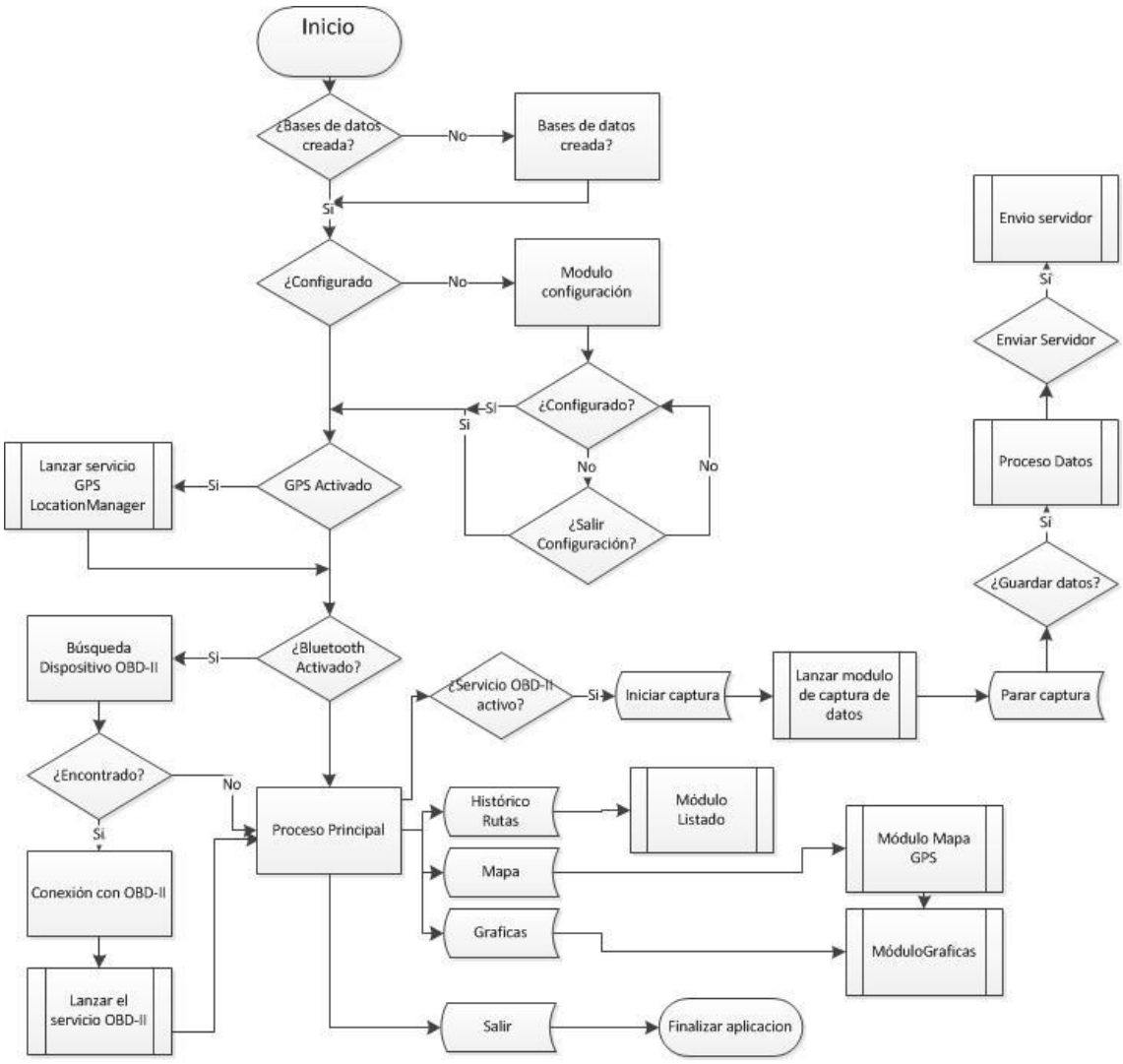


Figura 5.2.1.1: Diagrama de flujo del módulo principal.

En el diagrama de la figura 5.2.1.1, se describe el flujo del módulo principal donde se hacen una serie de comprobaciones referentes al dispositivo móvil. Se ha decidido que **NO** sea necesario que el Bluetooth y el GPS estén activos, para así poder disponer de las demás funcionalidades de la aplicación; en el caso de que solo el GPS esté activo, las únicas capturas que se realizan serán las coordenadas de éste y viceversa.

Por otro lado, utilizando la información de la configuración de la aplicación, en concreto la dirección MAC del dispositivo emparejado al cual el dispositivo se debe conectar (ELM327 Bluetooth), el dispositivo móvil realiza una conexión de prueba. En caso de que no sea posible la ejecución de la actividad principal, se notificará al usuario mediante un mensaje de error indicándole si desea reintentarlo. Si la conexión se efectúa correctamente el icono de bluetooth cambiara a color azul.

```
BluetoothSocket btSocket;
BluetoothDevice                          btDevice                          =
btAdapter.getRemoteDevice(MyOptions.getString("devicePaired", null));

try {
      btSocket =
      btDevice.createRfcommSocketToServiceRecord(UUID.fromString("0000
      1101-0000-1000-8000-00805F9B34FB"));
      btSocket.connect();
      btSocket.close();
      return true;
} catch (IOException e1) {
      return false;
}
```

Figura 5.2.1.2: Código de conexión de bluetooth con la interfaz ELM327.

El código que se muestra en la figura 5.2.1.2 permite realizar la prueba de conexión con el interfaz ELM327 Bluetooth. Como se observa, mediante la primitiva "btAdapter.getRemoteDevice" y la dirección MAC que se encuentra en la configuración de la aplicación, obtenemos el nodo al que queremos realizar la conexión (ELM327 Bluetooth). Android solo proporciona conexiones RFCOMM, por lo que, para obtener un socket Bluetooth, hacemos uso de la primitiva "btDevice.createRfcommSocketToServiceRecord(UUID)", siendo necesario especificar el UUID (Universal Unique Idenfifier) del servicio RFCOMM. Una vez se obtiene el socket bluetooth se puede intentar realizar la conexión con el interfaz ELM327.

5.2.2 Esquema base de datos.

En la aplicación de Android se ha creado una base de datos, *drivingStylesDB*, que consta de 2 tablas: en la primera de ellas, *ds_fich_datos,* generamos un registro o tupla por cada ruta registrada en la aplicación del móvil, y se registran todos los datos generales de la ruta, como la velocidad media, número de muestras registradas, hora inicio y fin de la misma, si ha sido enviada al servidor correctamente, etc. La otra tabla mencionada anteriormente es *ds_datos*, y registra los datos recibidos desde el OBD-II, como la velocidad, la posición del pedal del acelerador, las revoluciones del motor (todas ellas instantáneas), además de otros datos que, o bien se obtienen de primitivas del propio móvil (como la latitud y longitud, velocidad registrada a partir del GPS, las aceleraciones ejercidas en el móvil), o son datos calculados por la aplicación, como la aceleración del coche. La clave de esta tabla está relacionada con la tabla ds_fich_datos, mediante una relación de 1 a N como podemos ver en la figura 5.2.2.1.

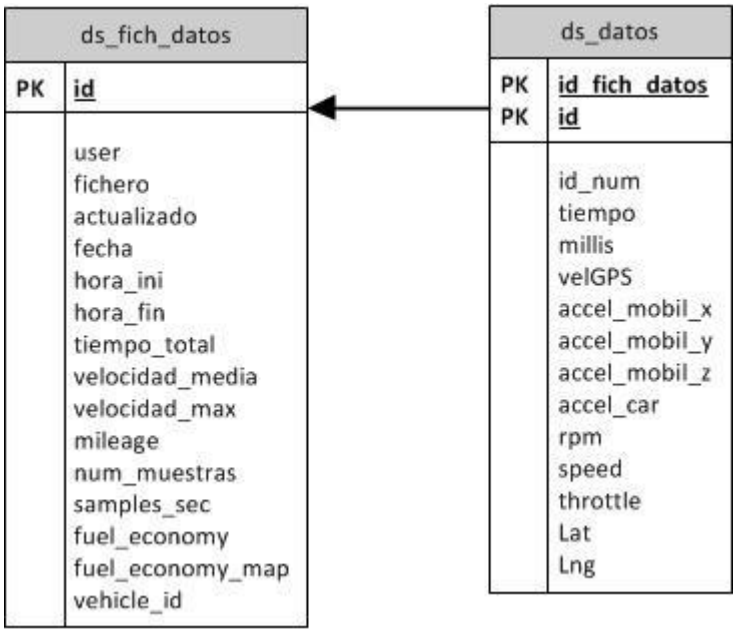


Figura 5.2.2.1: Esquema de la base de datos de la aplicación Android.

5.3 Módulo Envío Ruta.

A este módulo se accede bien desde el histórico de las rutas almacenadas o cuando dejamos de capturar datos.

Figura 5.3.1: Pantalla envío de datos.

La pantalla informativa visualiza la información de cabecera de la ruta seleccionada.

- Fecha de la captura de datos.
- Hora inicio.
- Hora finalizada la ruta.
- Tiempo total de la captura.
- Velocidad media.
- Velocidad máxima.
- Kilómetros totales realizados.
- Muestras capturadas.
- Muestras tomadas por segundo.

Este módulo incluye las siguientes funcionalidades:

Gráficas: Muestra las gráficas de la velocidad, las aceleraciones y las revoluciones por minuto de la ruta.

Mapa: Muestra el itinerario seguido por el coche, si disponemos conexión a internet veremos el mapa y la ruta tomada.

Borrar: Elimina la ruta de la base de datos.

Enviar: Lanza el modulo de envío de ruta al servidor, el cual leerá los datos pertenecientes a la ruta y generará el fichero XML, enviándolo al servidor. El usuario y el password no se envían con el fichero, siendo elementos de autenticación en la conexión con el servidor, como vemos en la figura 5.3.2.

```
sendDataServerThread = new Thread() {
  public void run() {
       try{
        HttpParams httpparametros = new BasicHttpParams();
        HttpConnectionParams.setConnectionTimeout(httpparametros, 8000);
        HttpConnectionParams.setSoTimeout(httpparametros, 8200);

       final HttpClient httpclient = new DefaultHttpClient(httpparametros);
       final HttpPost httppost = new HttpPost(URL);

       MultipartEntity multipartContent = new MultipartEntity();

       File file = null;
       file = new File(directorio);

       multipartContent.addPart("user", new StringBody(usuario));
       multipartContent.addPart("password", new StringBody(UserPassword));

       multipartContent.addPart("file", new FileBody(file));
       httppost.setEntity(multipartContent);
```

Figura 5.3.2: Código conexión con el servidor DrivingStyles.

El fichero recibido en el servidor *DrivingStyles* se guarda en el directorio correspondiente del usuario, y también se genera un registro en la base de datos por cada muestra enviada para su posterior análisis.

5.3.1 Diagrama de Flujo.

La figura 5.3.1.1 muestra el flujo del módulo de envío de rutas. Desde aquí podemos acceder a ver el itinerario realizado en la ruta, acceder a las gráficas, eliminar la ruta y, como hemos comentado anteriormente, enviar los datos al servidor.

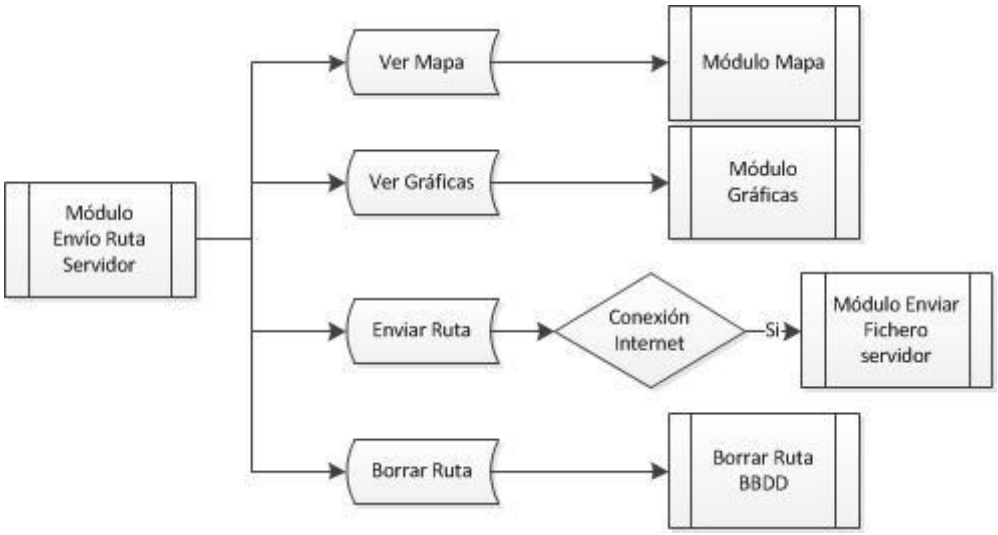


Figura 5.3.1.1: Diagrama de flujo del módulo Envío Ruta al Servidor.

La figura 5.3.1.2 muestra la generación del fichero XML que se envía al servidor; este fichero no se guarda en la aplicación, sino que es generado cada vez que es enviado. El sistema nos devuelve un mensaje de error en caso que se produzca algún fallo, o un mensaje de confirmación desde el servidor si el fichero es recibido correctamente.

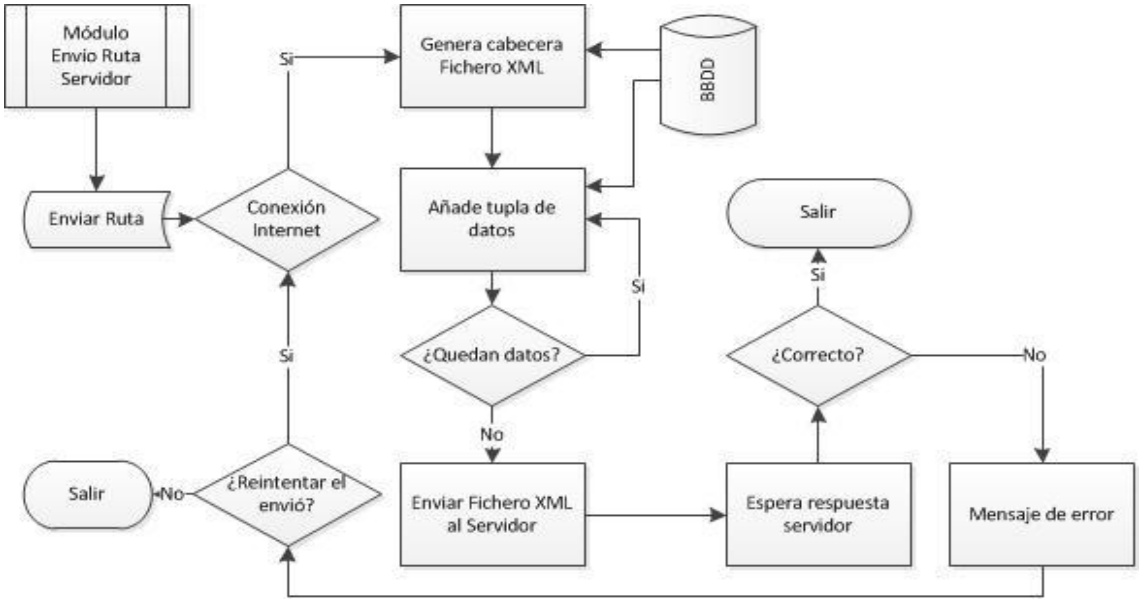


Figura 5.3.1.2: Diagrama de flujo del módulo Genera XML.

5.3.2 Estructura del Fichero.

El fichero que se envía al servidor es un fichero XML, y se corresponde con la estructura de la base de datos presentada en la sección 5.2.2. La cabecera del fichero, que corresponde con la etiqueta *USUARIO*, es el registro correspondiente a la ruta en la tabla *ds_fich_datos*, y los atributos dentro de la etiqueta *DATOS* corresponden con los datos capturados de la aplicación de dicha ruta (figura 5.3.2.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<DRIVINGSTYLES>
<USUARIO>
        <NOMBRE>usuario1</NOMBRE>
        <NOMBREFICH>ruta1</NOMBREFICH>
        <FECHA>2012-04-18</FECHA>
        <HORAINI>05:44:15</HORAINI>
        <HORAFIN>05:57:33</HORAFIN>
        <TIEMPOTOTAL>13:18</TIEMPOTOTAL>
        <VELOCIDADMEDIA>49.164</VELOCIDADMEDIA>
        <VELOCIDADMAX>131</VELOCIDADMAX>
        <MUESTRAS>1354</MUESTRAS>
        <FUELECO>0</FUELECO>
        <FUELECOMAP>0</FUELECOMAP>
        <VEHICLEID/>
        <SAMPLESSEC>2</SAMPLESSEC>
        <MILEAGE>10.8</MILEAGE>
</USUARIO>
<DATO lng="-0.344557" lat="39.5309" velGPS="22" accelMobilZ="0.07" accelMobilY="0.12"
accelMobilX="0.02" accelCar="2.86" throttle="22" speed="26" rpm="1855"
millis="541041" tiempo="05:53:16" id_num="1"/>
<DATO lng="-0.344557" lat="39.5309" velGPS="22" accelMobilZ="0.1" accelMobilY="0.16"
accelMobilX="0.09" accelCar="1.07" throttle="28" speed="27" rpm="2098"
millis="541615" tiempo="05:53:17" id_num="2"/>
……
</DRIVINGSTYLES>
```

Figura 5.3.2.1: Fichero XML enviado al servidor.

Un punto importante a destacar es que los datos no se envían como etiquetas, sino como atributos de una etiqueta llamada *DATO.* Con esto conseguimos reducir mucho el tamaño del fichero XML, así como el tiempo de envío al servidor, con el consiguiente ahorro. Por ejemplo, en un registro de ruta de 2500 muestras (2 por segundo y 21 minutos de registro) el tamaño es de 507 Kbyte. Cada línea del fichero corresponde con una muestra de 150 caracteres y, en el caso de que cada dato se envíe dentro de una etiqueta, el fichero pasaría a tener unos 400 caracteres por registro (el número de caracteres varía dependiendo de la muestra), casi 2,5 veces mas grande, y el tamaño del fichero XML pasaría a ser de unos 1200 Kbyte.

5.4 Módulo Gráficas.

Las gráficas que podemos visualizar, tanto en tiempo real como con rutas previamente almacenadas, están realizadas con las librerías estándar de Android.

```
android.graphics.Color;
android.graphics.PointF;
```

Se añaden, además, otras librerías GPL[16].

```
com.androidplot.xy.BoundaryMode;
com.androidplot.xy.LineAndPointFormatter;
com.androidplot.xy.LineAndPointRenderer;
com.androidplot.xy.XYPlot;
```

Si el modelo de móvil lo permite, se puede realizar también zoom para visualizar toda la gráfica o parte de ésta utilizando la pantalla táctil del dispositivo.

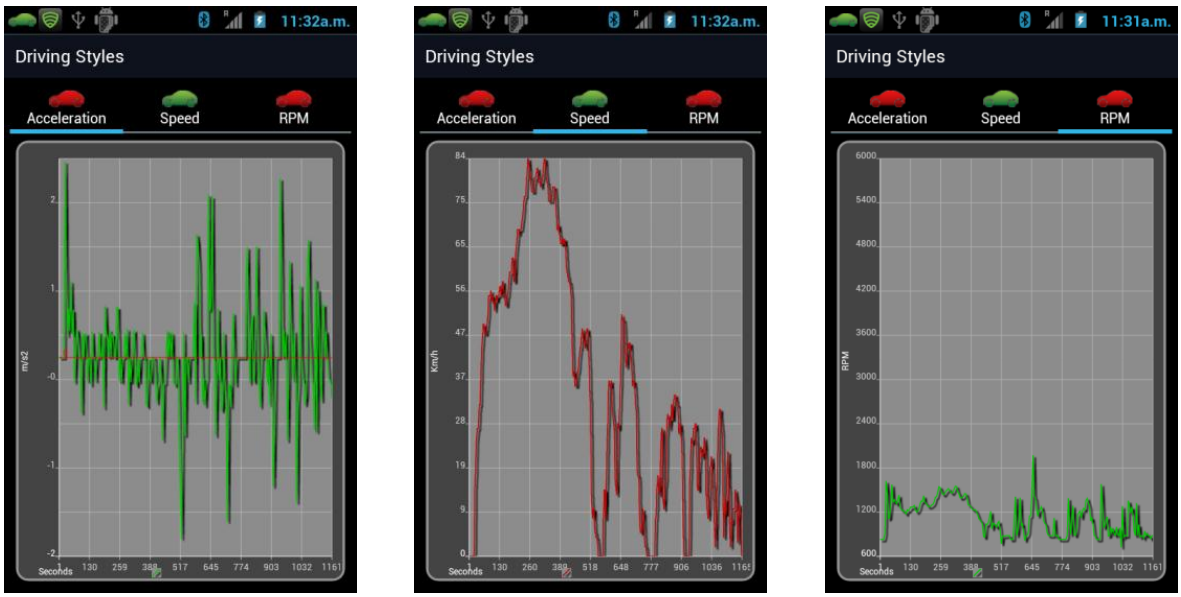


Figura 5.4.1: Pantallas de gráficas de la aceleración, la velocidad y las revoluciones del motor.

Se han seleccionado para representar gráficamente la aceleración, la velocidad y las revoluciones del motor. Hemos escogido representar gráficamente estos tres parámetros porque son los más relevantes, y los que utilizaremos en el entrenamiento de la red neuronal.

5.4.1 Diagrama de Flujo.

Accedemos al modulo de visualización de gráficas desde dos pantallas distintas de la aplicación. Si partimos desde el modulo principal vemos las gráficas en tiempo real con una ventana de visualización de los últimos 10 segundos, la cual se desplaza cada segundo; la otra forma de acceder a éstas es desde el módulo de envío ruta, y aquí las gráficas que visualizaremos son las registradas y grabadas anteriormente.

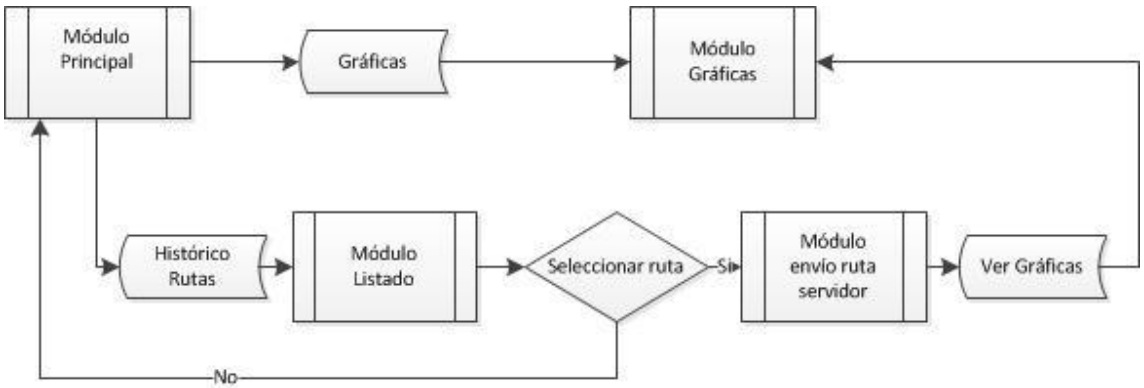


Figura 5.4.1.1: Diagrama de flujo del módulo Gráficas.

5.5 Módulo Mapa.

Este módulo permite visualizar en el mapa la posición de coordenadas GPS obtenida por el dispositivo móvil. Se dibujan las coordenadas GPS utilizando las siguientes APIs de Google para representación de mapas.

```
com.google.android.maps.GeoPoint;
com.google.android.maps.MapActivity;
com.google.android.maps.MapController;
com.google.android.maps.MapView;
com.google.android.maps.OverlayItem;
```

Figura 5.5.1: Pantalla mapa.

Mediante un icono de un coche verde se indica el inicio de la ruta, y con otro icono rojo la posición actual del vehículo; también la ruta se representa utilizando diferentes colores dependiendo de la velocidad del vehículo.

- Menor de 50km se representa en verde.
- Entre 51km y 90km se representa en amarillo.
- Entre 91Km y 120km se representa en naranja.
- Más de 120km se representa en rojo.

5.5.1 Diagrama de Flujo.

Como en el caso del módulo de gráficas, accedemos a la visualización del mapa desde dos pantallas distintas de la aplicación. Si partimos desde el módulo principal vemos la posición actual del vehículo y la ruta generada hasta el momento. A través del módulo de envío ruta se visualiza el mapa completo de toda la ruta.

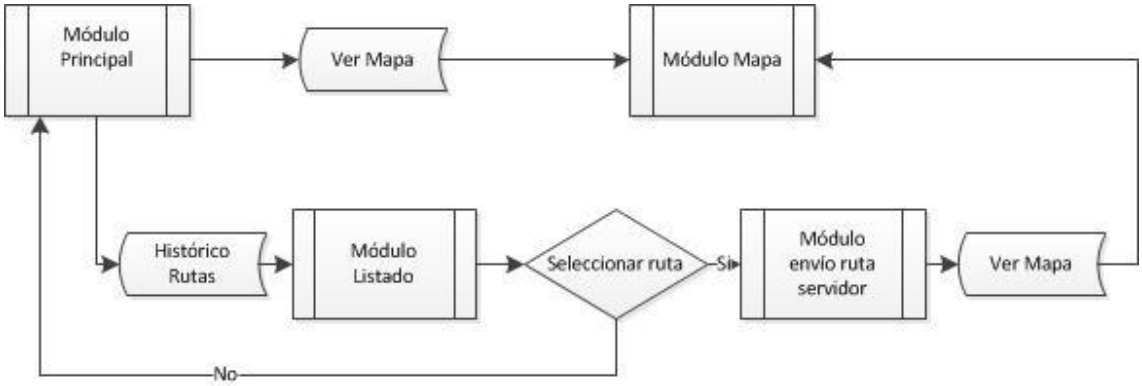


Figura 5.5.1.1: Diagrama de flujo del módulo mapa.

5.1 Módulo Listado de Rutas.

Desde el módulo principal accedemos al listado de las rutas grabadas previamente, mediante diferentes iconos se indica si la ruta ha sido enviada ya al servidor o no.

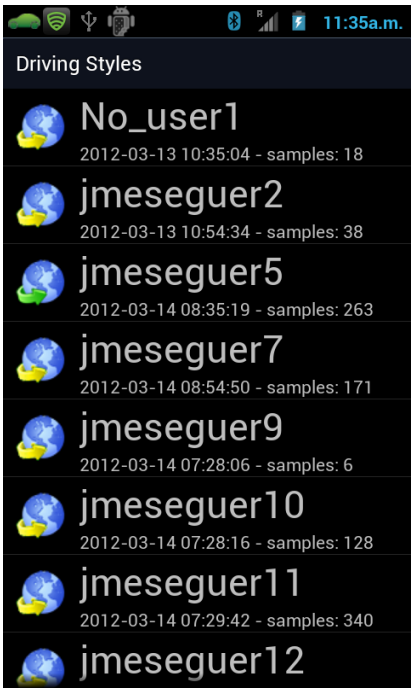


Figura 5.1.1: Pantalla listado de rutas capturadas.

5.1.1 Diagrama de Flujo.

Seleccionado una ruta llegamos al módulo de envío de ruta al servidor como vemos en la figura 5.1.1.1.

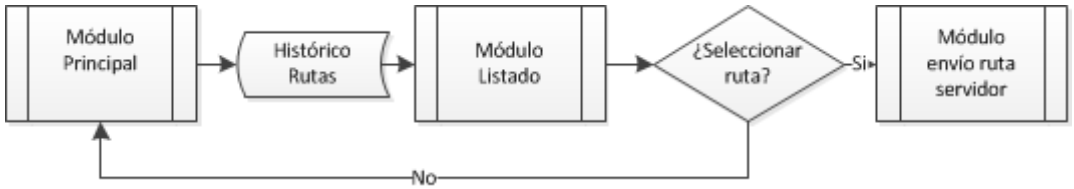


Figura 5.1.1.1: Diagrama flujo del módulo listado.

6 APLICACIÓN PORTAL WEB DRIVINGSTYLES.

El segundo gran bloque de este trabajo de fin de Máster es el componente relativo al portal Web. Para su desarrollo se ha utilizado un servidor de software libre, HTTP Apache, y como gestor de contenido Joomla. Al principio del proyecto se pensó en realizar la parte web sin utilizar ningún gestor de contenido, pero los resultados no fueron los esperados y el coste temporal de esta parte del trabajo se alargaba. Se pensó entonces en utilizar un gestor de contenido en el cual, mediante el uso del recurso wrapper nos desvinculamos de la parte de la presentación, centrándonos en el problema.

Para adecuar la presentación del portal a la que se deseaba se modificó la hoja de estilos y la plantilla del portal. También se añadieron campos a las tablas genéricas para el correcto funcionamiento del proyecto.

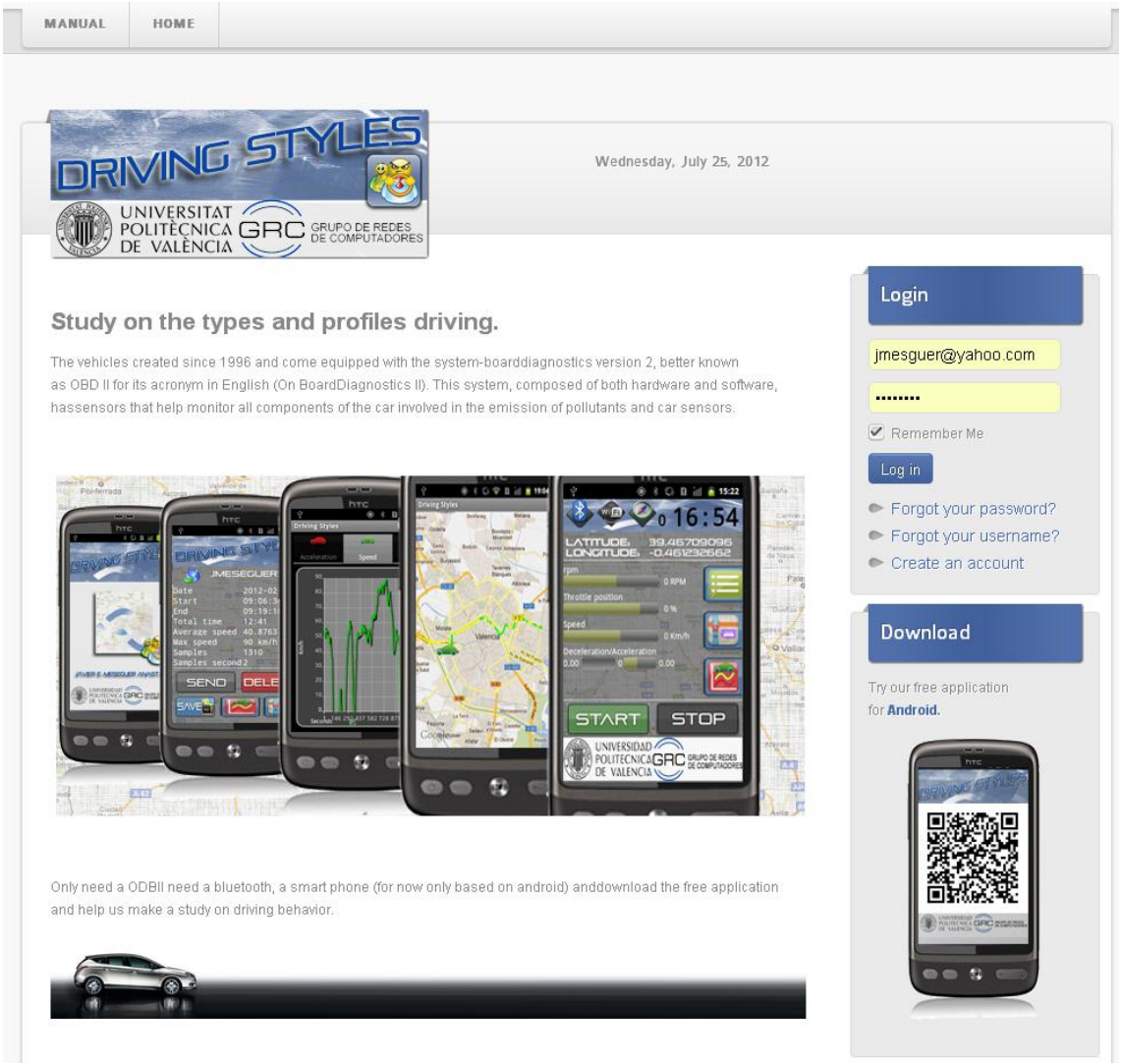


Figura 6.1: Portal Web, Pantalla inicio.

6.1 Usuario.

Una vez que el usuario se registra ya puede acceder a la sección usuario. En ella se registran una serie de datos que son importantes, sobre todo para futuros estudios de minería de datos. Respecto a los datos de tipo personal, los más interesantes son el sexo, la edad, y otros datos concernientes al vehículo que utiliza, como marca, modelo, tipo de combustible y aceleración de 0-100 teórica del vehículo. Este último es importante para normalizar los datos del estudio. Por último, un tercer bloque permite que el conductor indique cual cree que es su comportamiento delante del volante.

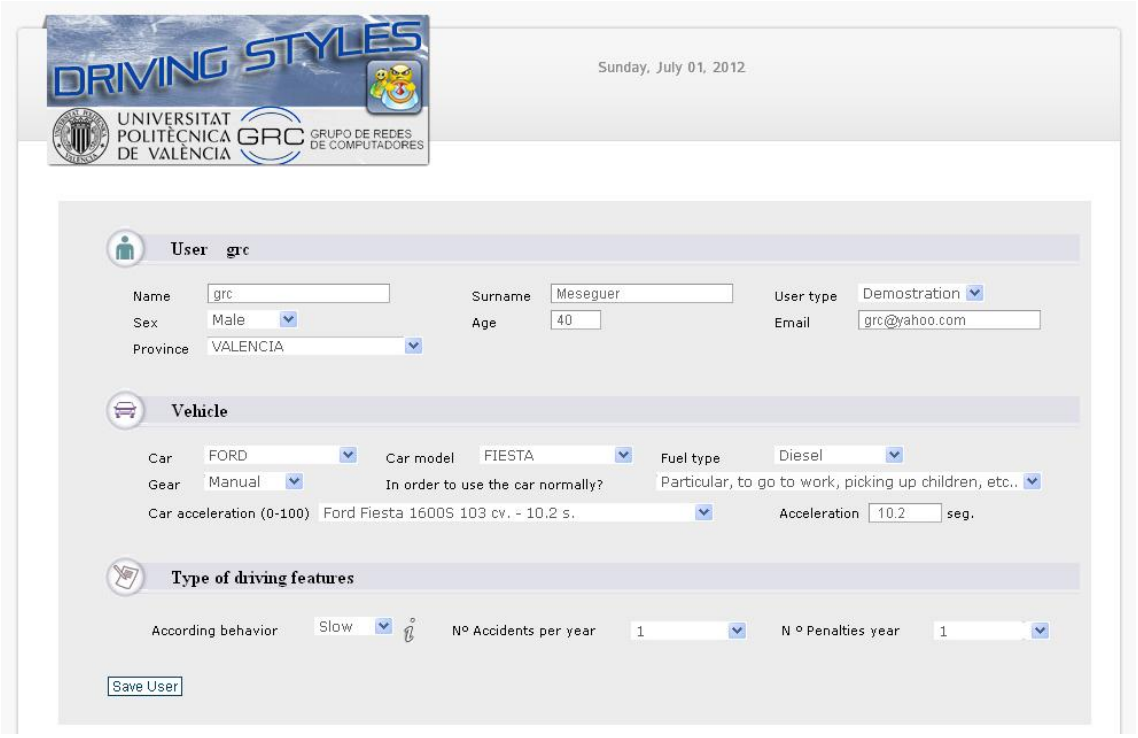


Figura 6.1.1: Portal Web, apartado usuario.

6.2 Rutas.

En el apartado Rutas el usuario puede acceder a todas las rutas que ha enviado al portal. En la primera tabla tenemos las rutas que están en la base de datos con el nombre, fecha, hora de inicio, hora fin, muestras enviadas, tiempo total, velocidad media y kilómetros realizados. Seleccionando una ruta aparece en una segunda tabla los datos que capturamos con el dispositivo móvil como la velocidad, la aceleración, las revoluciones por minuto del motor y las coordenadas GPS de cada una de las muestras tomadas.

En un tercer bloque tenemos el itinerario realizado por la ruta seleccionada, dependiendo de la velocidad del vehículo se dibuja con un color u otro (de la misma forma que en la aplicación Android).

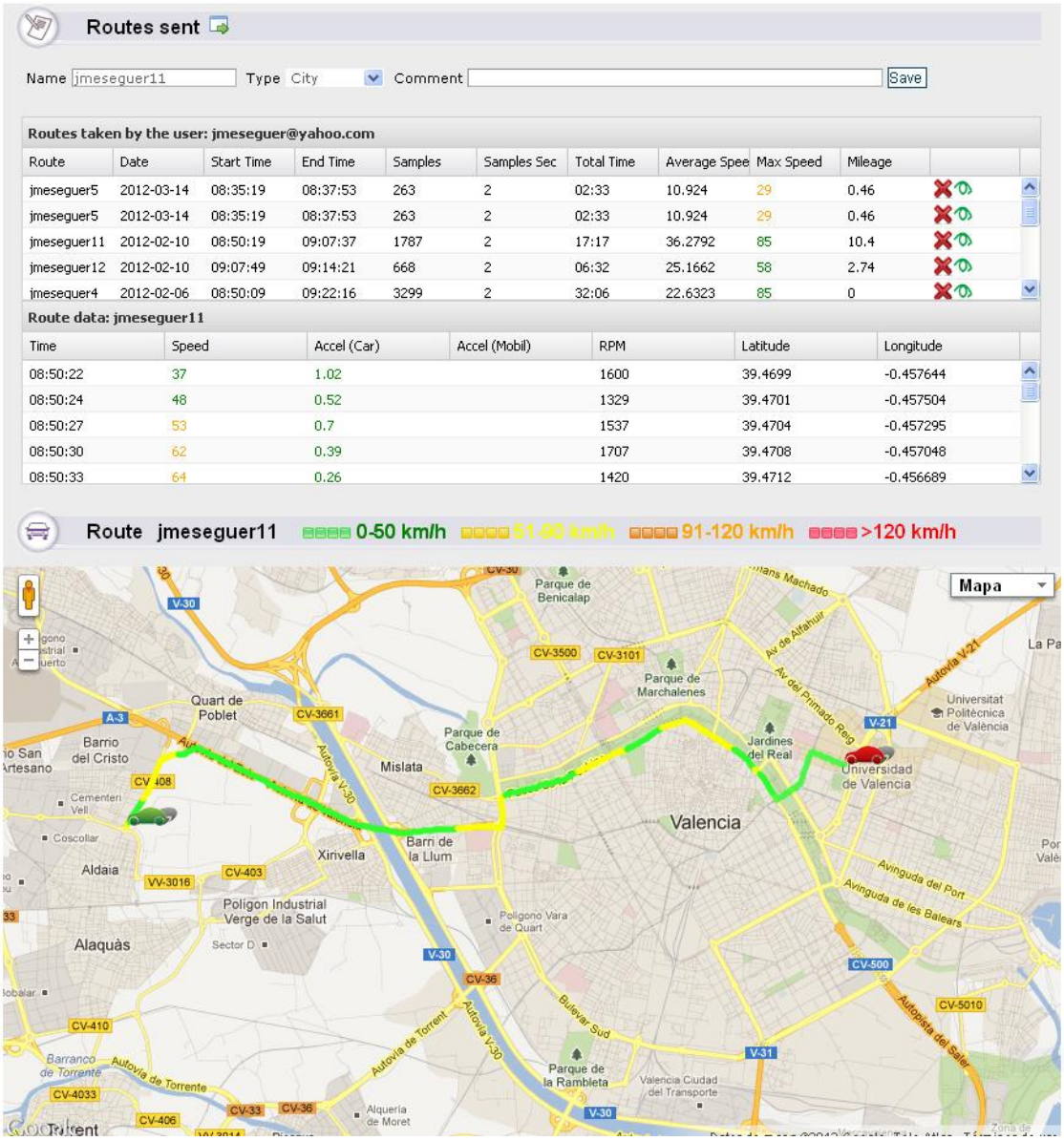


Figura 6.2.1: Portal Web, apartado rutas.

6.3 Estadísticas.

En esta sección el usuario puede ver los datos que ha capturado su dispositivo móvil.

Dispondrá de las gráficas de velocidad, aceleración del vehículo, aceleración del móvil y las revoluciones por minuto del motor. Además, las dos últimas gráficas muestran los resultados que le devuelve la red neuronal (en la sección séptima se verán los detalles da la RNA) cuando se realiza una segmentación del recorrido según características de la vía por la que ha circulado, y también de su estilo de conducción en cada tipo de vía.

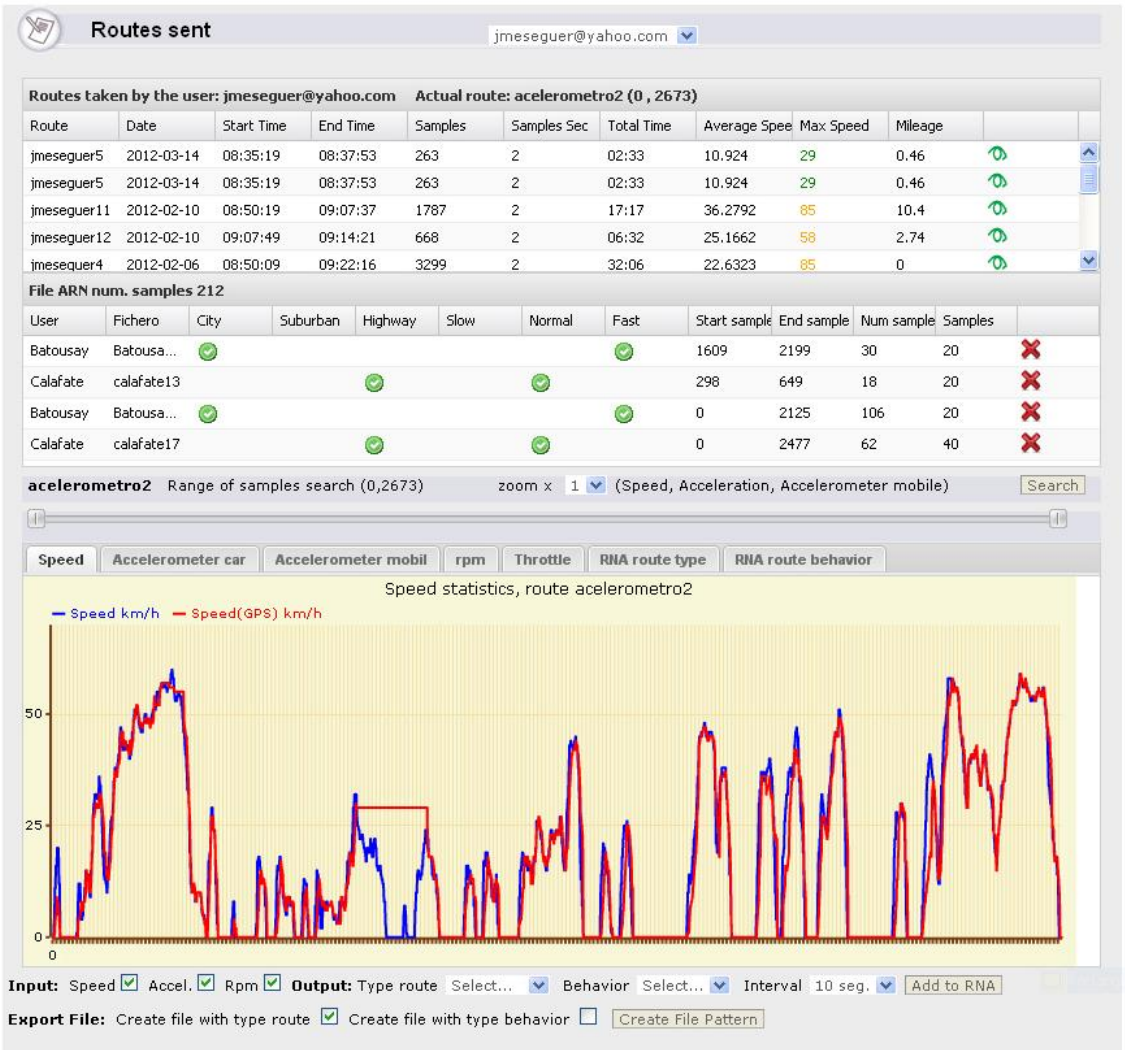


Figura 6.3.1: Portal Web, apartado estadísticas (velocidad del vehículo).

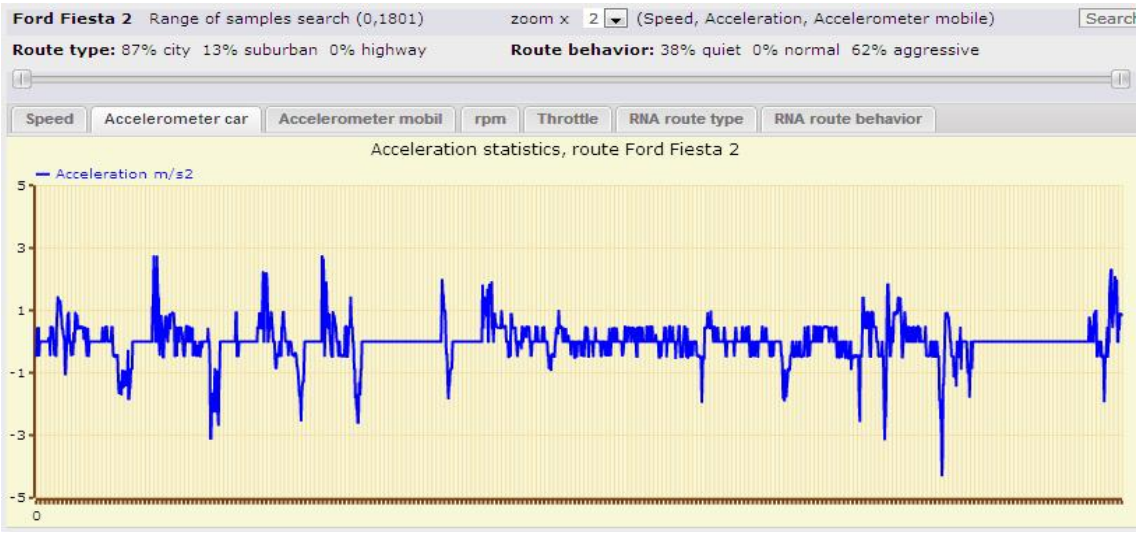


Figura 6.3.2: Portal Web, apartado estadísticas (aceleración del vehículo).

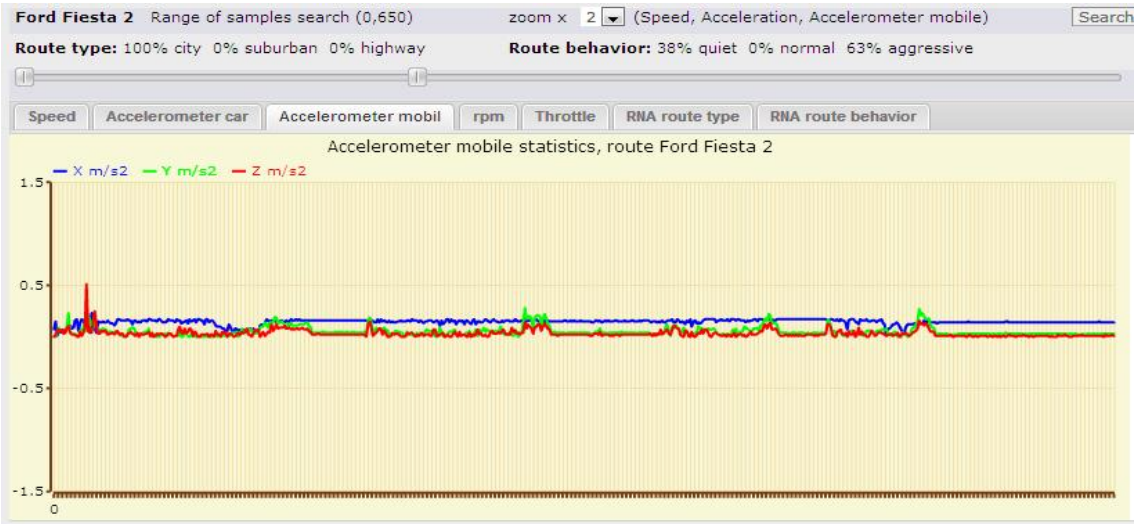


Figura 6.3.3: Portal Web, apartado estadísticas (aceleración del móvil).

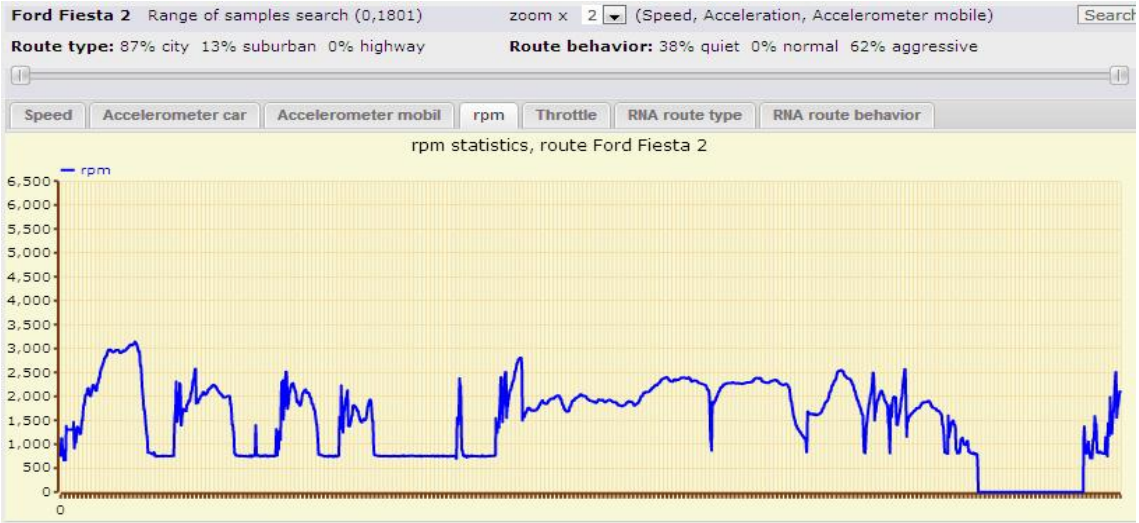


Figura 6.3.4: Portal Web, apartado estadísticas (revoluciones por minuto del motor).

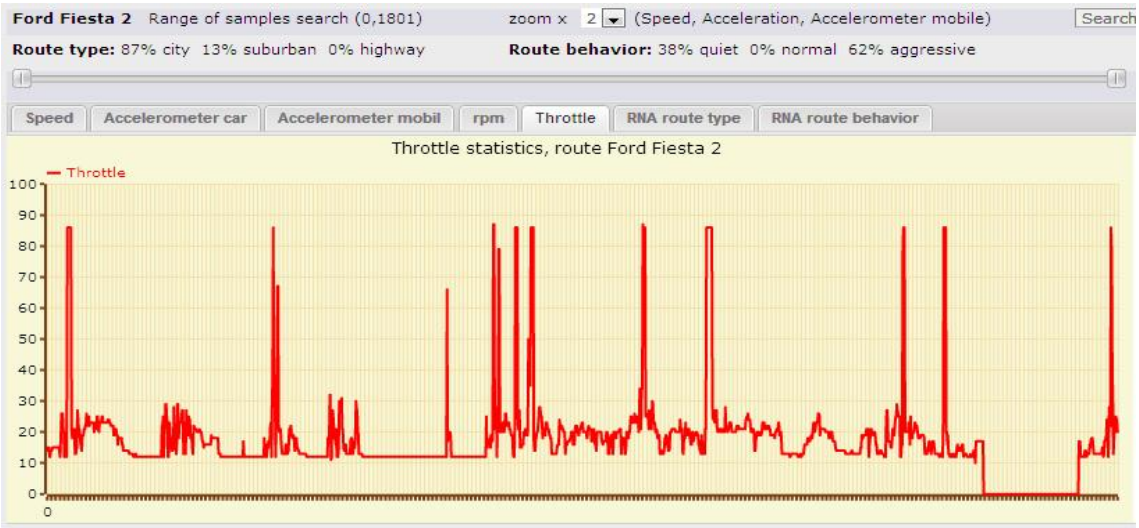


Figura 6.3.5: Portal Web, apartado estadísticas (posición pedal aceleración).

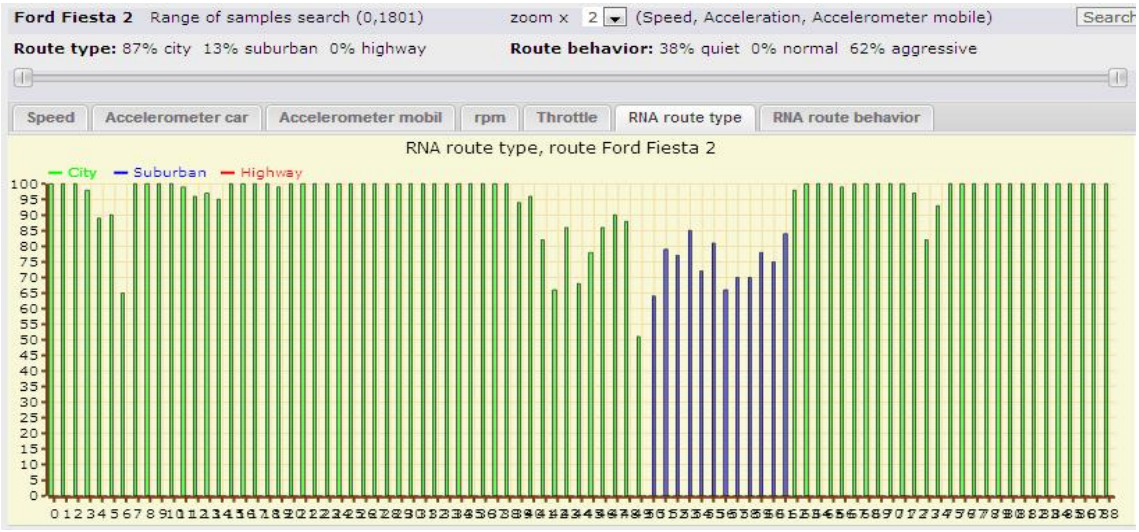


Figura 6.3.6: Portal Web, apartado estadísticas (tipo de ruta).

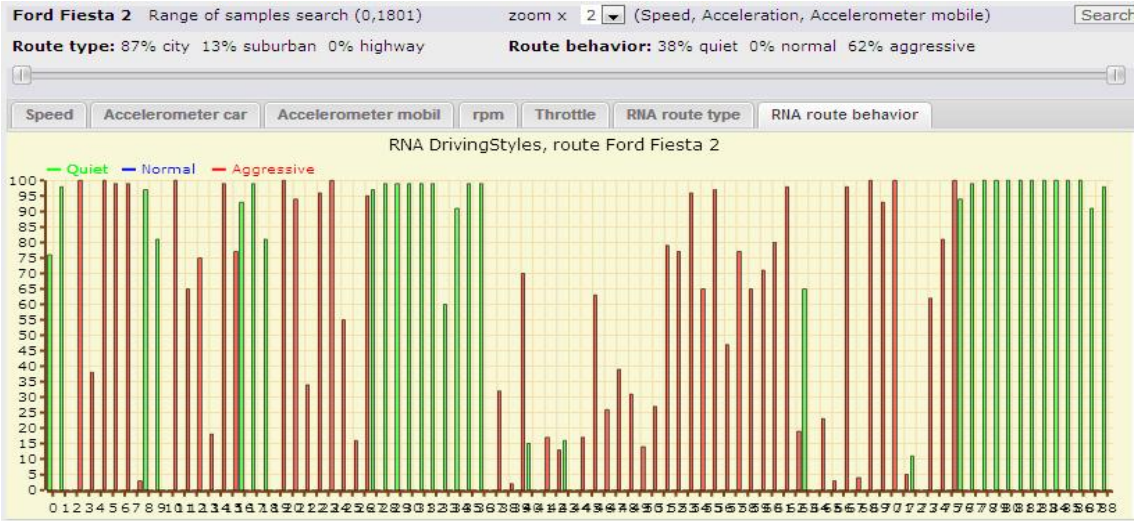


Figura 6.3.7: Portal Web, apartado estadísticas (tipo de comportamiento).

6.4 Generación de ficheros test y validación de la RNA.

El objetivo de este componente web es generar ficheros para poder realizar un entrenamiento supervisado de la red neuronal. Para ello se dispone de un modo de acceso como "superusuario" al portal. Éste dispone de todas las rutas que hay registradas en el sistema, y puede seleccionar las rutas, o tramos de éstas, que tengan un perfil claro respecto al tipo de vía en que circula el vehículo y el estilo de conducción del conductor. Esta información es etiquetada y añadida a la base de datos, realizándose la misma operación con tantas rutas como se crea necesario para el entrenamiento supervisado de la red (ver figura 6.4.1).

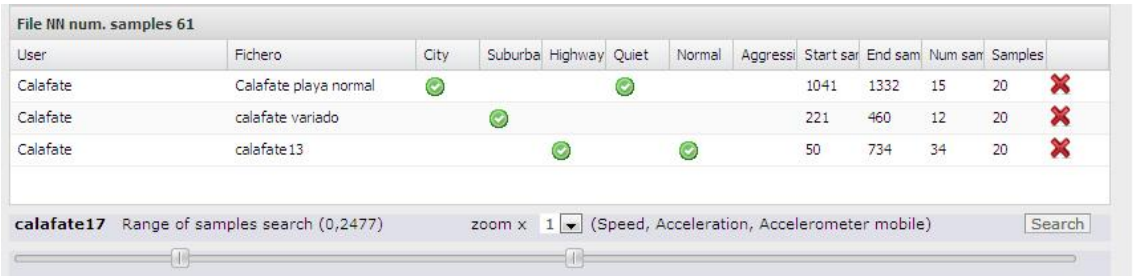


Figura 6.4.1: Listado de las rutas que se utilizan en el entrenamiento de la red neuronal.

Posteriormente, cuando tenemos suficientes muestras se genera el fichero de entrenamiento y validación de la red neuronal, y se selecciona qué tipo de entrada tendrá la red.

Figura 6.4.2: Exportación del fichero de entrenamiento de la red neuronal.

La estructura del fichero que reconoce la aplicación JavaNNS es el siguiente:

```
SNNS pattern definition file V3.2
generated at 09-07-2012 15:13:15


No. of patterns : 104
No. of input units : 6
No. of output units : 3

# Input pattern 1:
0.433333 0.0333333 0.029918 0.353793 0.434195 0.00764575
# Output pattern 1:
0 0 1
# Input pattern 2:
0.470833 0.0544862 0.117623 0.471716 0.449088 0.0472043
# Output pattern 2:
0 0 1
# Input pattern 3:
0.283333 0.0408248 0.0118852 0.365729 0.276748 0.0296485
# Output pattern 3:
. . . . . .
0 0 1
```

Figura 6.4.3: Fichero generado para realizar el entrenamiento de la red neuronal.

6.5 Estructura bases datos.

La base de datos de la parte web es conceptualmente similar a la de la aplicación del dispositivo móvil: tenemos dos tablas donde guardamos los datos que recibimos de la aplicación Android – ds_fich_datos y ds_datos (idéntica a la aplicación anterior) –, y dos tablas más con una estructura muy similar – ds_fich_datos_RNA y ds_datos_RNA – donde guardamos los datos ya formateados (velocidad, aceleración y rpm normalizados) para poder generar los ficheros de validación y de test de la Red Neuronal. Se guarda la media y la desviación estándar de cada uno de los datos anteriores, agrupados de 20 en 20.

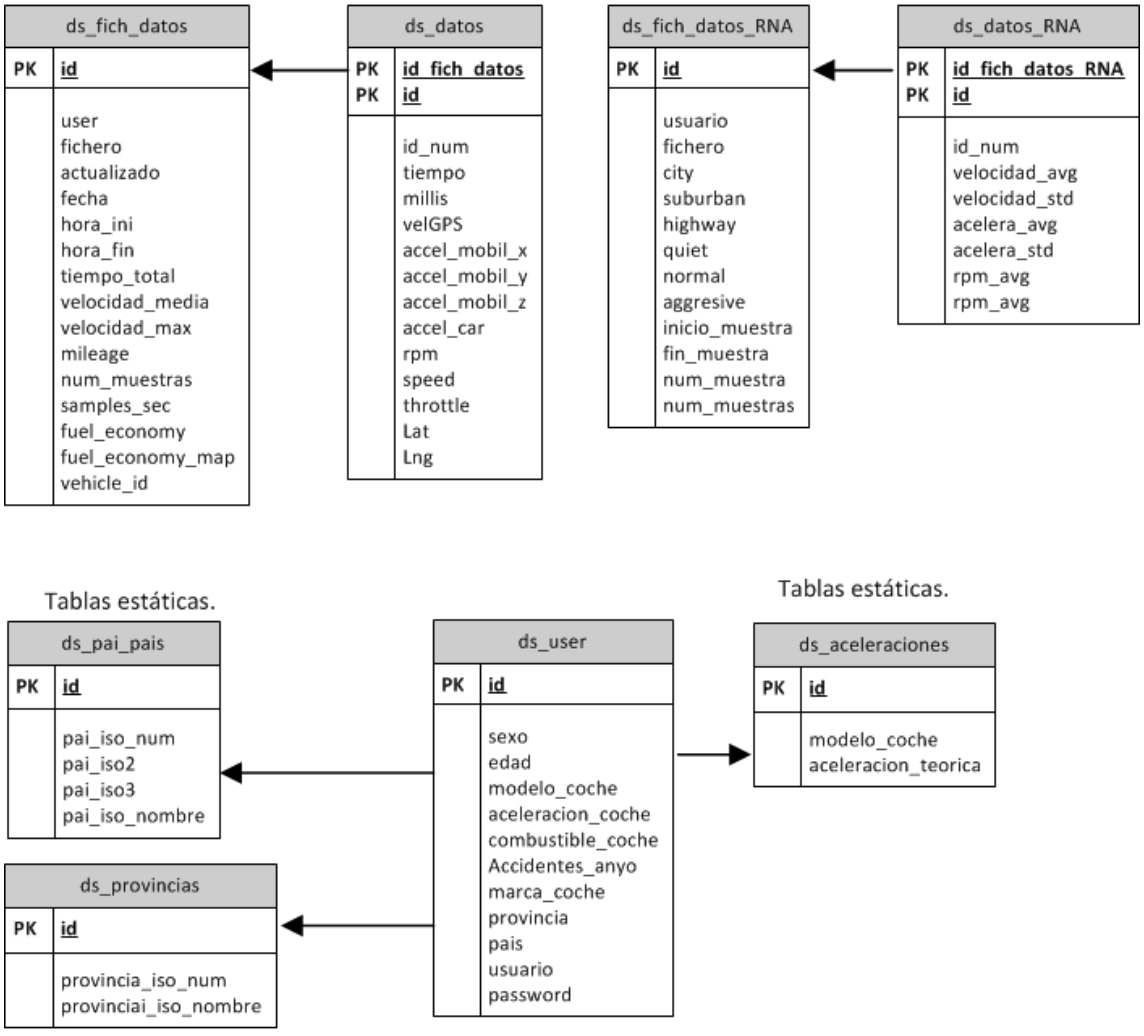


Figura 6.5.1: Diagrama de la base de datos de la aplicación web.

Parte II - Análisis Sobre Redes Neuronales.

7 REDES NEURONALES.

7.1 Introducción a las Redes Neuronales.

Las Redes Neuronales Artificiales (RNA) [20] están inspiradas en la biología, en la naturaleza las neuronas reciben señales (entradas) de otras neuronas a través de conexiones sinápticas que pueden ser excitantes o inhibidoras. En función de las señales recibidas, una neurona envía a su vez una señal a otras neuronas por medio del axón. Una neurona no hace nada, a menos que la influencia colectiva de todas sus entradas alcance un nivel de umbral. Siempre que se alcanza tal umbral, la neurona produce una salida, que consiste en un pulso que se desplaza del cuerpo de célula, por el axón, hasta las ramas de éste, y en este caso se dice que la neurona de dispara. Una neurona con este comportamiento, se dice que es un dispositivo de todo o nada.

Las Redes Neuronales Artificiales (RNA) tratan de emular este comportamiento. Según Haykin, S. [13] "Una red neuronal es un procesador paralelo distribuido que tiende a almacenar conocimiento experimental y lo utiliza en sus cálculos. Se asemeja al cerebro en dos aspectos:

1. El conocimiento se adquiere a través de un proceso de aprendizaje.
2. Existe fuerte conexión entre neuronas conocido como pesos sinápticos utilizado para almacenar conocimiento."

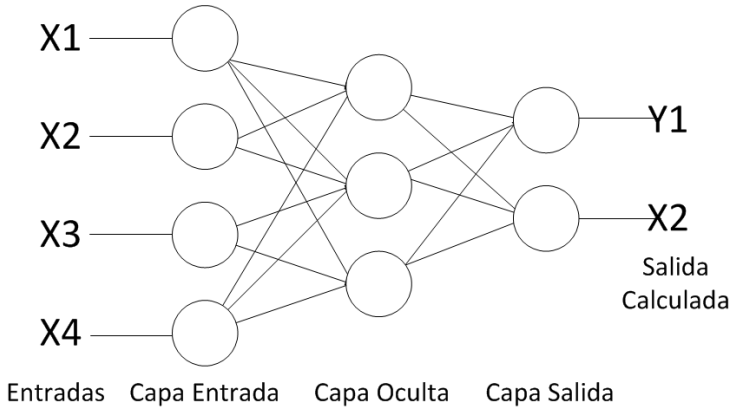


Figura 7.1.1: Esquema de neurona artificial.

Las redes de neuronales artificiales consisten en neuronas como la que se muestra en la figura 7.1.1. La neurona simulada se ve como un nodo conectado con otros mediante enlaces que corresponden a conexiones *axón-sinapsis-dendrita*. Cada célula (unidad de proceso), suministra un valor a su salida, y este valor se propaga a través de la red de conexiones unidireccionales hacia otras células de la red. Asociada a cada conexión hay un peso w_{ij}, que determina el efecto de la célula j-ésima sobre la célula i-ésima, que determina la influencia de un nodo sobre otro, es el producto de la salida de la neurona que influye por el peso del enlace que los conecta. Un peso positivo

grande corresponde a una excitación fuerte, y un peso negativo pequeño corresponde a una inhibición débil. La función de Activación es aquella en la cuál cada nodo combina las influencias separadas que recibe en sus enlaces de entrada en una influencia global.

Cada célula (unidad de proceso), suministra un valor a su salida, este valor se propaga a través de la red de conexiones unidireccionales hacia otras células de la red.

Una sola función de activación pasa la suma de los valores de entrada a través de una función de umbral para determinar la salida del nodo, que puede ser 0 ó 1, dependiendo de si la suma de las entradas está por encima o por debajo del valor de umbral utilizado por la función de umbral de nodo.

7.2 Entrenamiento.

La principal característica de las Redes Neuronales es su capacidad para aprender de sus entradas, y mejorar su tasa de acierto a través del aprendizaje. Una Red Neuronal aprende mediante un proceso interactivo de ajustes de sus pesos sinápticos y niveles de sesgo (bias).

La secuencia de eventos que ocurren durante el aprendizaje es la siguiente:

1. Estimulación de la red Neuronal por sus entradas.
2. La red neuronal sufre cambios en sus parámetros como resultado de dicha estimulación.
3. La red neuronal responde de manera diferente a las nuevas entradas debido a los cambios que ocurrieron en su estructura interna.

Se define como algoritmo de aprendizaje al conjunto de reglas bien definidas para la solución de un problema de aprendizaje. Existe una gran variedad de algoritmos de aprendizaje (backpropagation para redes feedforward, Elman, Real-Time Learning Algorithm, etc.), teniendo cada uno sus propias ventajas. Los algoritmos de aprendizaje difieren entre sí en la forma como se formulan los cambios en los pesos sinápticos.

En el trabajo de fin de Máster realizado el algoritmo que se ha escogido es el de *backpropagation*, examinando el tipo de problema que tenemos que resolver vemos que es de tipo clasificación, partiendo de unos datos de entrada, que en nuestro caso son la velocidad, la aceleración, y las revoluciones del motor obtenemos como salida el tipo de vía que circula y el estilo de conducción.

7.2.1 Redes Neuronales Supervisadas y No Supervisadas.

Las redes neuronales se clasifican comúnmente en términos de sus correspondientes algoritmos o métodos de entrenamiento:

- Redes de pesos fijos (para estas redes no existe ningún tipo de entrenamiento).
- Redes supervisadas.
- Redes no supervisadas.

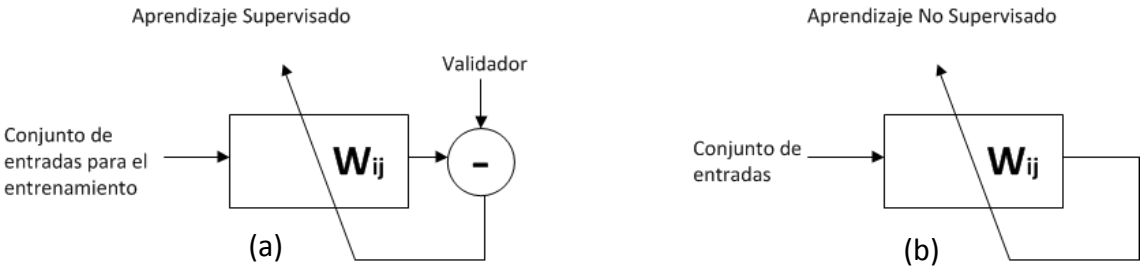


Figura 7.2.1.1: Entrenamiento supervisado (a) y no supervisado (b).

Reglas de entrenamiento Supervisado:

Los datos para el entrenamiento están constituidos por varios pares de patrones de entrenamiento de entrada y de salida. El hecho de conocer la salida implica que el entrenamiento se beneficia la supervisión de un maestro. Dado un nuevo patrón de entrenamiento los pesos serán adaptados como vemos en el aprendizaje supervisado de la figura 7.2.1.1 (a).

Reglas de entrenamiento No Supervisado:

El conjunto de datos de entrenamiento consiste sólo en los patrones de entrada. Por lo tanto, la red es entrenada sin el beneficio de un maestro. La red aprende a adaptarse basada en las experiencias recogidas de los patrones de entrenamientos anteriores, como vemos en el aprendizaje no supervisado de la figura 7.2.1.1 (b).

El entrenamiento de la red neuronal en el proyecto ha sido con reglas de entrenamiento supervisado, a los patrones de entrada se les asigna un patrón de salida (indicado por nosotros a la hora de creación de los ficheros de prueba y validación).

7.3 Backpropagation.

Backpropagation [18] es un tipo de red de aprendizaje supervisado que da muy buenos resultados en los problemas de tipo clasificación, como es el del presente proyecto. Emplea un ciclo propagación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, éste se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Los valores de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red reciben una señal de error que nos de su contribución al error total. A partir de aquí se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento.

A medida que se entrena la red, las neuronas de las capas intermedias se organizan y aprenden a reconocer distintas características de los datos de entrada. Después del entrenamiento, las neuronas responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica a reconocer, y para la cual han sido entrenadas.

7.4 Aplicaciones utilizadas.

La aplicación que se ha utilizado para la creación y el entrenamiento de las redes neuronales en el trabajo de fin de Máster ha sido JavaNNS [9], la cual es una versión en java del programa SNNS[12] de la universidad de TÜBINGEN [17]. Tiene una interfaz gráfica amigable además de un manual on-line [10] para empezar a trabajar de modo eficaz. Los datos para entrenar y validar la red neuronal se pasan mediante un fichero *.pat (en la propia web hay un directorio con ejemplos) de texto plano y de fácil generación, como hemos visto anteriormente.

En nuestro sistema el problema surge a la hora de exportar posteriormente el modelo generado para codificarlo en el portal web, ya que JavaNNS no tiene implementada la generación de código de ningún tipo, pero sí permite exportar ficheros con la configuración de la red neuronal extraída; empleando posteriormente la utilidad *snns2.exe* [12] conseguimos generar código C *(Ver anexo2).*

8 Aspectos metodológicos del entrenamiento de las redes neuronales ruta y comportamiento.

8.1 Variables escogidas.

Una de las etapas del preprocesado de los datos consiste en seleccionar de entre todas las posibles variables de entrada a la red neuronal que hemos considerado en un principio, un subconjunto mínimo de estas variables imprescindible para obtener las dos redes neuronales que nos muestre el tipo de ruta por la que se circula y el modo de conducción.

Obviamente si disminuimos demasiado el número de variables de entrada nos encontraremos con redes neuronales muy manejables, con pocas entradas y ciclos de aprendizaje rápidos pero a costa de disminuir el rendimiento de la red por haber prescindido de variables importante para la clasificación. En el extremo contrario, podemos incluir demasiadas variables de entrada, algunas fuertemente correlacionadas y, por tanto, redundantes, lo que resulta en redes a las que proporcionamos toda la información de que disponemos pero que presentan ciclos excesivamente largos y a veces un rendimiento bajo porque se le hace llegar información de forma indiscriminada.

Entrada 1	Velocidad media.
Entrada 2	Desviación estándar velocidad.
Entrada 3	Aceleración media.
Entrada 4	Desviación estándar aceleración.
Entrada 5	Rpm media.
Entrada 6	Desviación estándar rpm.

Tabla 8.1.1: Variables de entrada de las redes neuronales.

En la práctica, el subconjunto no es mínimo sino un compromiso entre un número manejable (no excesivamente grande) de variables y un rendimiento de la red aceptable, en el caso del proyecto de la tesina del Máster nos vemos supeditados a las variables que podemos obtener de la *Electronic Control Unit* (ECU) del vehículo tres de esas posibles variables que hemos escogido para entrenar la red neuronal son la velocidad, la aceleración del vehículo y las revoluciones del motor (la media y la desviación estándar de las tres variables), en todos los vehículos que hemos utilizado para las pruebas en el proyecto se obtenían sin ninguna dificultad, otras variables como la posición del pedal del acelerador que se consideramos que nos proporcionaría información para el entrenamiento de la red neuronal, se descarto porque no todos las ECUs de los fabricantes devuelven este valor.

8.2 Normalización de los datos.

Los datos de cada parámetro de entrada se normalizan entre 0 y 1. Esta normalización deberá hacerse con todo el rango de valores posibles, es decir el valor mínimo y máximo de cada variable del dominio de entrada que se utilizan en la creación de los ficheros de entrenamiento y test.

La fórmula utilizada para normalizar los parámetros de entrada es la siguiente:

$$x^{p\prime} = \frac{x^p - \min(x)}{\max(x) - \min(x)}$$

Siendo $x^{p\prime}$ el valor normalizado de la variable de entrada x del patrón p, x^p el valor original de la variable x de ese patrón p, y min(x), max(x) los valores mínimo y máximo de esa variable de entrada, es decir los valores mínimo y máximo de la columna correspondiente a la variable x.

Uno de los motivos para realizar esta normalización es que la activación de la neurona de salida de la red neuronal multicapa sólo puede alcanzar valores comprendidos entre 0 y 1. Al normalizar los datos de esta manera todos los valores estarán dentro del mismo rango y los resultados obtenidos por los diferentes modelos podrán compararse directamente.

Una vez tenemos normalizados las variables de entrada ya podemos generar los ficheros de validación y de test. Como podemos ver en la figura 8.2.1 estos ficheros tienen una estructura de texto plano, con una disposición de campos concreta:

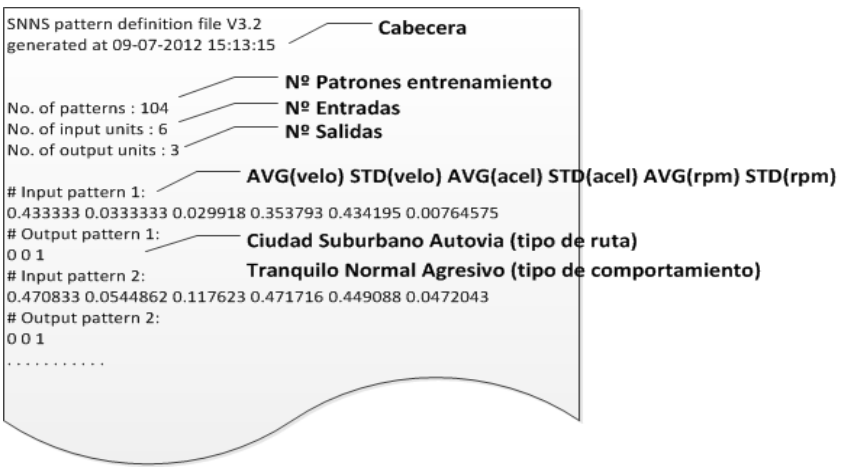


Figura 8.2.1: Estructura del fichero de entrenamiento y validación de la red neuronal.

En primer lugar tenemos la cabecera, con la versión del fichero y la fecha de generación de éste (útil para llevar un control sobre los ficheros generados desde la web).

Seguidamente tenemos tres líneas que nos definen los campos del fichero en cuestión, el primer campo nos indica el número de pares de entrada que tiene el fichero, el segundo campo cuantas variables de entrada hemos definido la red (capa de entrada) y por último el número de salidas (capas de salida), que en el proyecto son 6 y 3 respectivamente.

Por último y hasta el final del fichero tenemos todos los pares de entradas y salidas definiendo las entradas como *Input pattern* y el número identificador de ese patrón de entrada, en la línea siguiente cada una de las entradas separadas por un espacio en blanco, de igual manera se definen los parámetros de salida, *Output pattern*, con un 1 si corresponde al patrón deseado y con un 0 en caso contrario.

Si el fichero es de validación o prueba, para comprobar el correcto funcionamiento de la red neuronal, el campo de salida no existe (*Output pattern*) obligando a la red neuronal (previamente entrenada) que nos indique la respuesta a las entradas recibidas, este tipo de fichero nos sirve para verificar el correcto funcionamiento de la red neuronal que se ha entrenado, para ello nos basamos en el error medio para el conjunto de test:

$$E = \frac{\sum_{p=1}^{m} |d^p - y^p|}{\mathrm{m}}$$

Donde $|d^p - y^p|$ es el error absoluto para el patrón p, es decir el valor absoluto de la diferencia entre la salida deseada y la salida obtenida, m es el número de patrones y el sumatorio correspondería a la suma de los errores absolutos para todos los patrones.

8.3 Configuración.

Los pasos para generar las dos redes neuronales que aplicaremos después para caracterizar la vía y el comportamiento del conductor son los siguientes:

Primero tenemos que crear la red neuronal vacía, definiendo el número de entradas que en nuestro caso son seis (velocidad media y su desviación estándar, aceleración media y su desviación y la rpm media y su desviación estándar). En ambas redes neuronales, un mayor número de nodos ocultos permite mejorar la tasa de aciertos, pero tiene como efecto negativo el aumentar el tiempo de respuesta.

En nuestro proyecto, las tres salidas posibles en la caracterización del tipo de vía son: ciudad, suburbano, y autovía y para la red neuronal que caracteriza el estilo de conducción del usuario las posibles salidas son: tranquilo, normal y agresivo; tenemos que obtener dos redes neuronales por separado que tendrán las mismas variables de entrada.

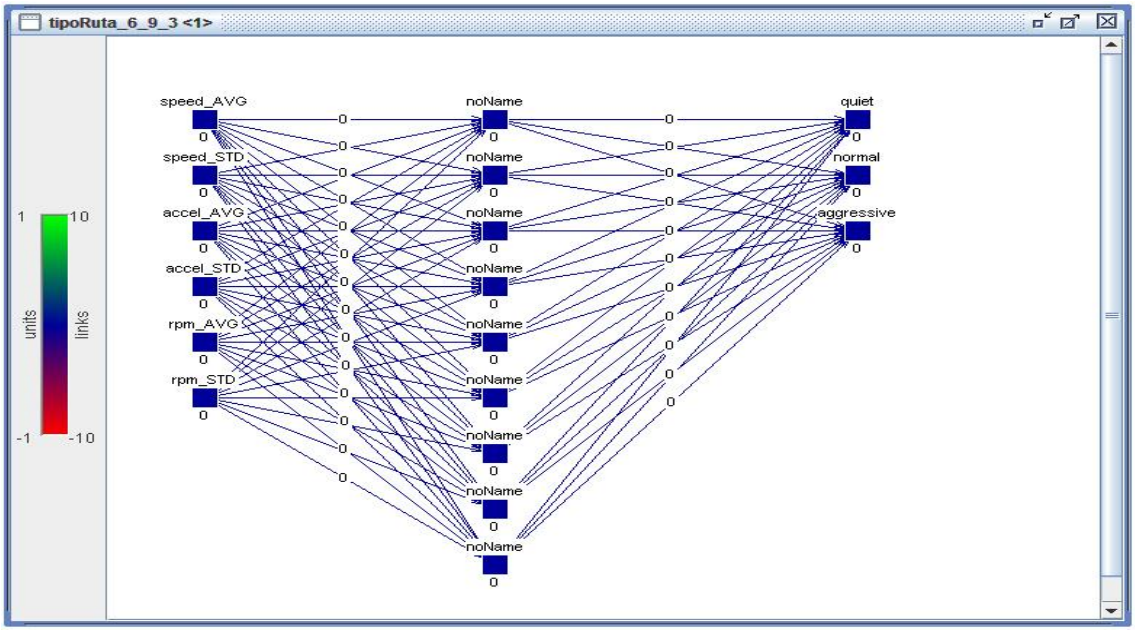


Figura 8.3.1: Red neuronal no entrenada para caracterizar comportamientos.

Una vez creada la red neuronal vacía con seis neuronas de entrada, nueve neuronas ocultas y tres neuronas de salida como se puede ver en la figura 8.3.1, escogemos la función de aprendizaje backpropagation [18] con la que pretendemos obtener buenos resultados para caracterizar las rutas y el estilo de conducción, se podía haber escogido otras funciones de aprendizaje como por ejemplo *backprop_momentum, hebbian, delta-rule, etc*. pero desde el principio nos decantamos por el ya que da muy buenos resultados en problemas de clasificación como el que estamos describiendo [18][19].

A continuación se generan los pesos de forma totalmente aleatoria para los enlaces de la red neuronal para ello utilizamos la función *ramdomize weights* para realizar una inicialización aleatoria de los pesos de la red neuronal, por defecto entre -1 y 1, en la figura 8.3.2 se puede ver la equivalencia del mapa de colores con sus respectivos valores de los pesos.

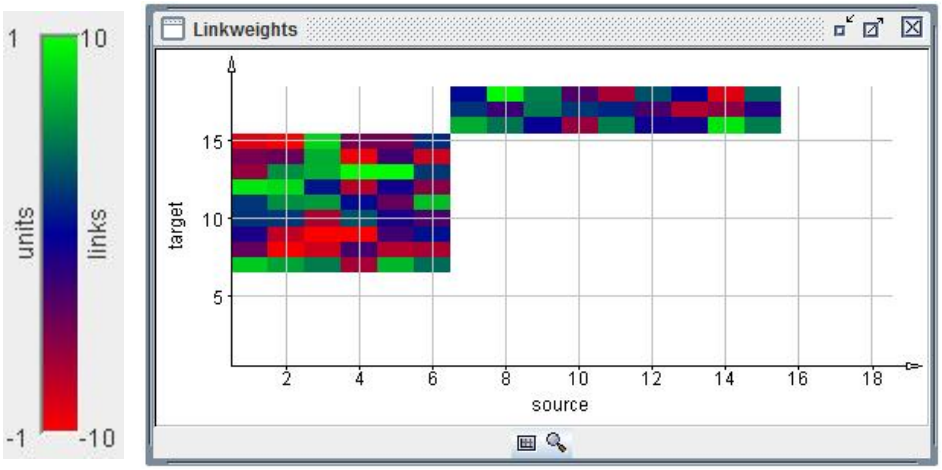


Figura 8.3.2: Pesos aleatorios de los enlaces de la red neuronal.

Como vemos en la figura anterior (figura 8.3.2) en el eje de abcisas se numeran las seis neuronas de la capa de entrada (1-6) y las nueve de la capa oculta (7-15). En el eje de ordenadas aparecen las de las capas ocultas (7-15) y las tres de salida (16-18).

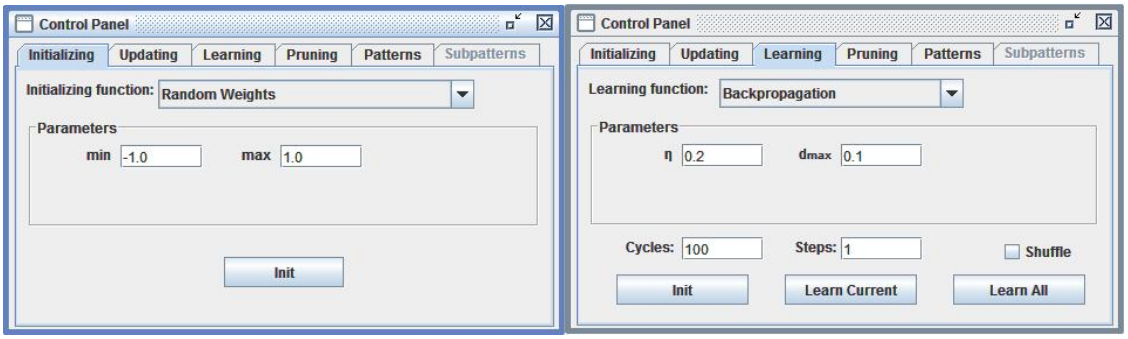


Figura 8.3.3: Inicialización de la red neuronal.

La red neuronal se guarda inicializada para asegurarnos que todas las pruebas que se realizan parten con los mismos pesos iniciales, evitando así que intervenga el azar, ya solo queda cargar los ficheros normalizados de validación y de test generados anteriormente desde la aplicación web para comenzar el entrenamiento.

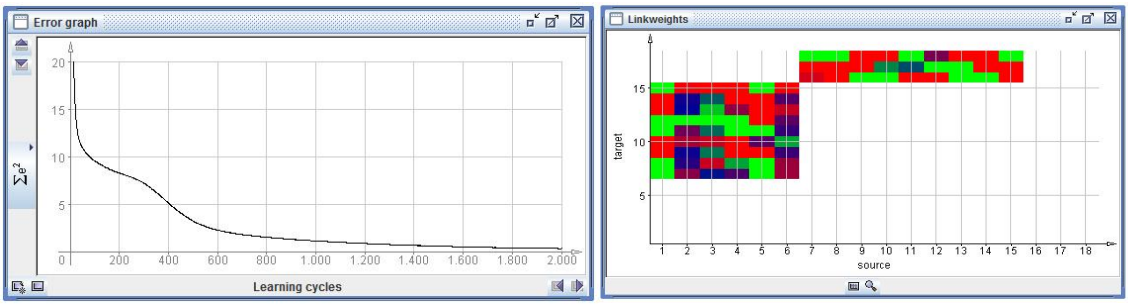
8.4 Entrenamiento

Una vez configurada la aplicación JavaNNS [9] empezamos el entrenamiento de la red neuronal, para que la aplicación vaya calculando el error cuadrático medio en cada iteración del aprendizaje necesitamos tener la ventana de errores visible y la de los pesos. Ajustamos la tasa de aprendizaje o razón de aprendizaje a 0.2 e iremos modificando para observar como repercute en el error de la red neuronal. A mayor tasa de aprendizaje, mayor es la modificación de los pesos en cada iteración, con lo que el aprendizaje es más rápido, pero por otro lado puede ocasionar oscilaciones indeseadas en la red (como posteriormente veremos en el apartado 8.4.2). También se irán ajustando el número de ciclos de aprendizaje que se efectúan en las pruebas.

8.4.1 Red Neuronal – Caracterización del tipo de vía.

8.4.1.1 Tasa de aprendizaje a 0.2.

Al iterar con una tasa de aprendizaje de 0.2, que es el valor que se usa por defecto, comprobamos cómo la red se queda estancada en un mínimo local con un error MSE de 0.021.

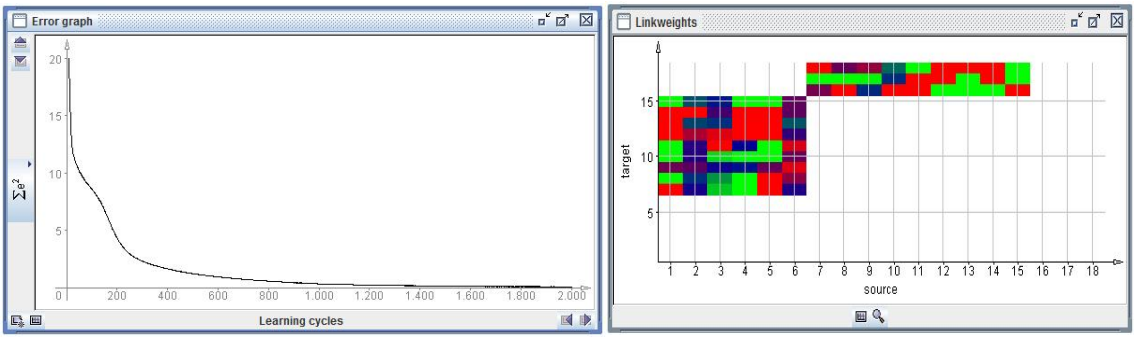


Gráfica 8.4.1.1: Convergencia y pesos en el entrenamiento, nº ciclos 2000 y n = 0.2.

Se puede observar que se obtiene un efecto curioso al pasar muy cerca de un mínimo local pero que finalmente se sobrepasa (sobre los 250 ciclos), llegando la red a converger, tal y como muestra la gráfica del error cuadrático medio.

8.4.1.2 Tasa de aprendizaje a 0.4.

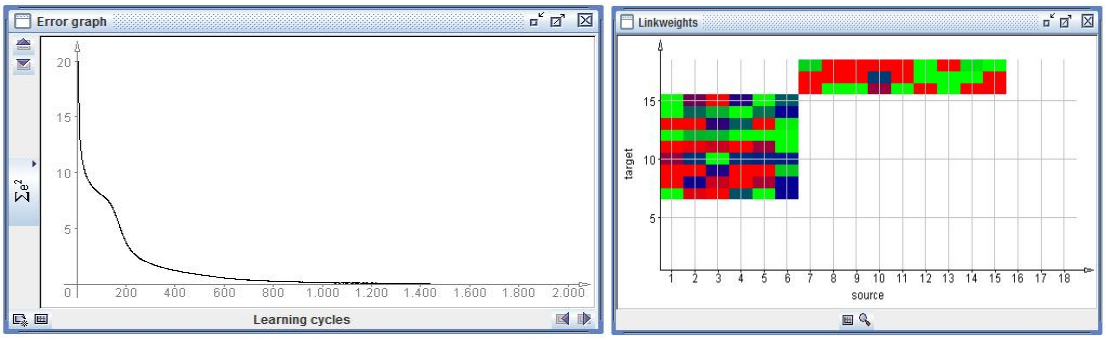
Si aumentamos la tasa de aprendizaje a 0.4, como es previsible, la red aprende mucho más rápidamente y converge en los 1200 ciclos, en 2000 ciclos el mínimo local tiene un error de MSE de 0.011.



Gráfica 8.4.1.2: Convergencia y pesos en el entrenamiento, nº ciclos 2000 y n = 0.4.

8.4.1.3 Tasa de aprendizaje a 0.6.

Si seguimos aumentando la razón de aprendizaje a 0.6 observamos que el mínimo local lo alcanza en 1400 ciclos y con un MSE de 0.009.



Gráfica 8.4.1.3: Convergencia y pesos en el entrenamiento, nº ciclos 2000 y n = 0.6.

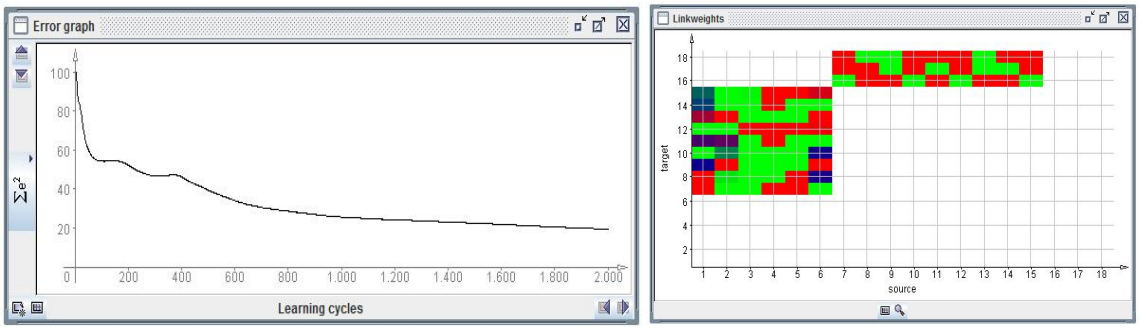
Cualquiera de las tres redes anteriores convergen de forma rápida dando un error muy pequeño, en la tesina del Máster se ha implementado esta última con una tasa de aprendizaje de 0.6, las otras dos redes primeras hubieran dado igualmente una buena solución a nuestro problema.

8.4.2 Red Neuronal – Caracterización del estilo conducción.

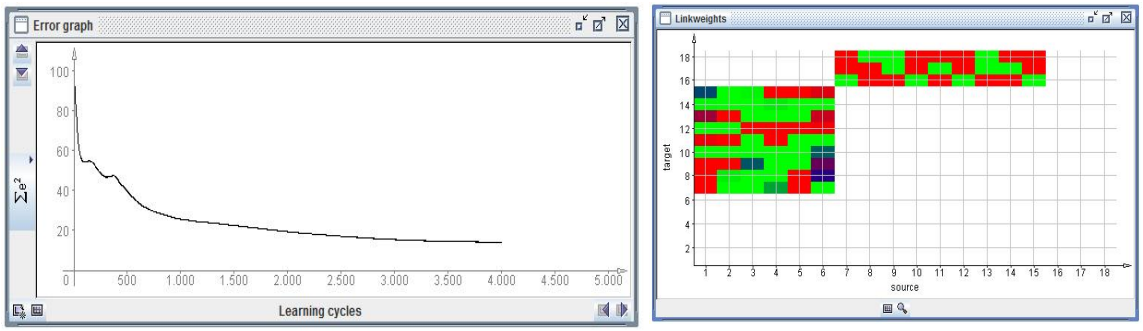
La caracterización del estilo de conducción es más compleja que el tipo de vía, no se llega al mínimo local tan rápidamente como en el caso anterior y el error de la red neuronal es mucho mayor.

8.4.2.1 Tasa de aprendizaje a 0.2.

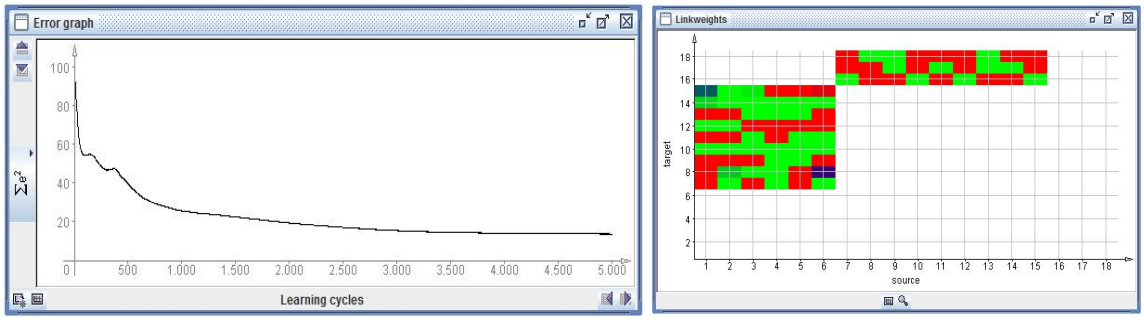
Como en el caso anterior empezamos con una tasa de aprendizaje de 0.2, pero se ve claramente que con 2000 ciclos no llegamos a un error mínimo que sea satisfactorio como vemos en la gráfica 8.4.2.1.



Gráfica 8.4.2.1: Convergencia y pesos en el entrenamiento, nº ciclos 2000 y n = 0.2.



Gráfica 8.4.2.2: Convergencia y pesos en el entrenamiento, nº ciclos 4000 y n = 0.2.

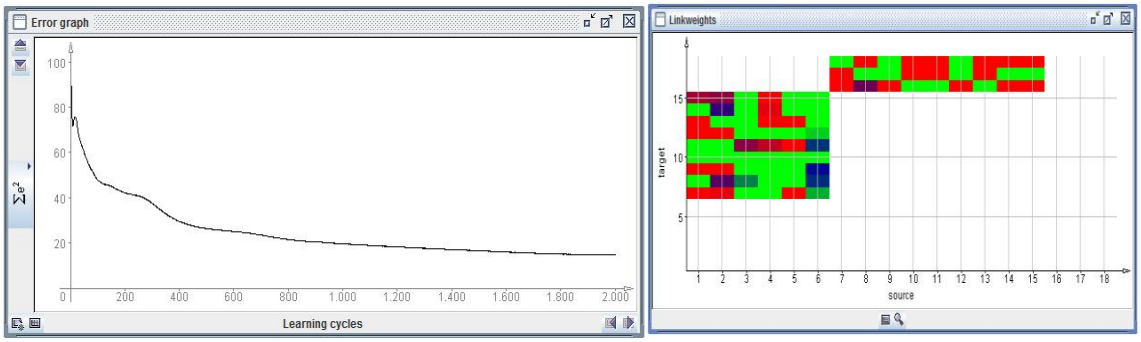


Gráfica 8.4.2.3: Convergencia y pesos en el entrenamiento, nº ciclos 5000 y n = 0.2.

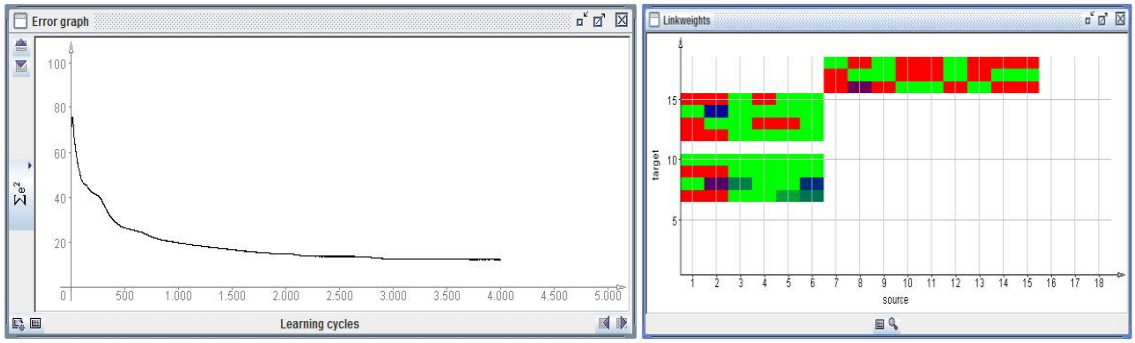
Tampoco se ve una mejora significativa en el error aumentando a 4000 y 5000 ciclos, la red no llega a converger de forma apreciable como en el caso de la red neuronal que nos implementa el tipo de vía, es este caso el error cuadrático medio es 0.54 y el error de la suma de cuadrados (SSE) es 13.43.

8.4.2.2 Tasa de aprendizaje a 0.4.

Ajustando la tasa de aprendizaje a 0.4 se ve que se mejora algo la convergencia, aunque no es demasiado significativa, en este caso el error cuadrático medio es 0.43 y el error de la suma de los cuadrados (SSE) es 11.06.



Gráfica 8.4.2.4: Convergencia y pesos en el entrenamiento, nº ciclos 2000 y n = 0.4.

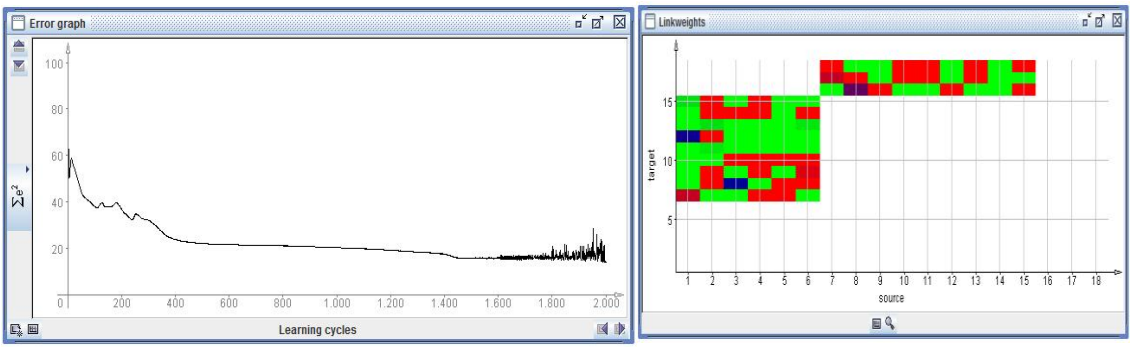


Gráfica 8.4.2.5: Convergencia y pesos en el entrenamiento, nº ciclos 4000 y n = 0.4.

Aumentar el tiempo de entrenamiento cambiando el número de ciclos a 4000 tampoco hace que converja mucho más la red neuronal.

8.4.2.3 Tasa de aprendizaje a 0.8.

Si aumentamos la tasa de aprendizaje a 0.8 vemos que la red neuronal desde el inicio del aprendizaje no se comporta normalmente, a partir de los 1500 ciclos empieza el error a oscilar, ello se debe a que los pesos varían demasiado rápidos haciendo la red neuronal sea inestable.

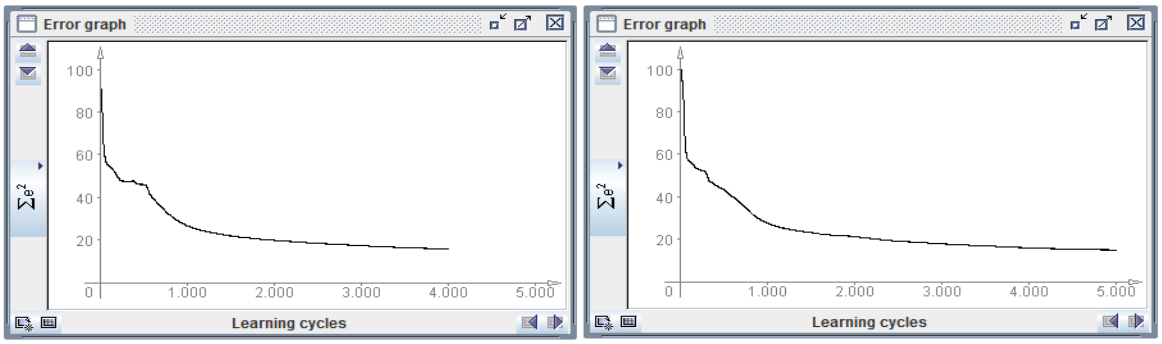


Gráfica 8.4.2.6: Convergencia y pesos en el entrenamiento, nº ciclos 2000 y n = 0.8.

Como es lógico el error cuadrático medio y el error de la suma de los cuadrados (SSE) se disparan, descartando totalmente esta implementación.

8.4.2.4 Elección de la red entrenada estilo de conducción.

Se realizaron otras pruebas para intentar minimizar el error que consistieron en mantener la tasa de aprendizaje constante (tasa a 0.2) y variando rango posible de los pesos que se asignan a la red pero los resultados fueron negativos como vemos en la figura 8.4.2.7.



Gráfica 8.4.2.7: Convergencia variando rango de los pesos [-0.5,0,5] y [-0.2,0,2], nº ciclos 4000 y n = 0.2.

Al final de las pruebas nos decantamos por utilizar la red neuronal entrenada con una tasa de aprendizaje de 0.4 que nos da el menor error obtenido de todas las pruebas realizadas, con un error cuadrático medio de 0.43 y el error de la suma de los cuadrados (SSE) es 11.06.

8.5 Implementación de la red obtenida.

Una vez entrenada satisfactoriamente la red neuronal hay que convertir el conocimiento obtenido en el código ejecutable. Para ello se dispone de la aplicación *snns2.exe* [12] que, pasándole la red entrenada como entrada, nos genera código en C (ver anexo 10.2). Este código esta integrado en el portal web desarrollado.

Con la red neuronal ya implementada en el portal web, cada vez que se selecciona una ruta o tramo de ruta, se cogen 20 muestras de la ruta (equivalente a 10 segundos) y se calcula la media y la varianza tanto de la velocidad, de la aceleración y de las revoluciones del motor, obteniendo seis parámetros que son las entrada de la red neuronal. Ésta nos devuelve el tipo de vía por la que se circula, y lo mismo ocurre con la red neuronal que estudia el estilo de conducción.

Gráficamente los parámetros de entrada a la red neuronal son:

1. Velocidad: Obtenemos la media y la varianza (figura 8.5.1).

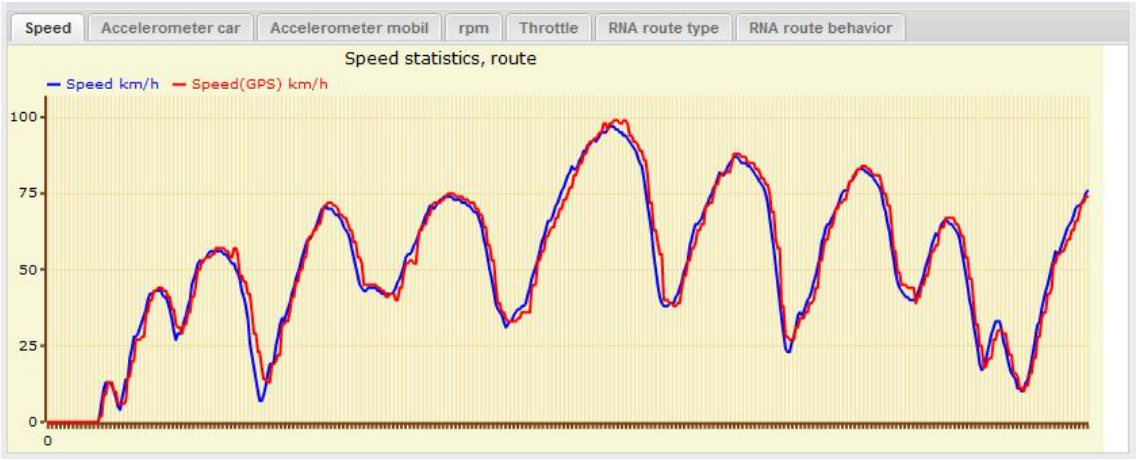


Figura 8.5.1: Gráfica velocidad.

2. Aceleración del vehículo: Obtenemos la media y la varianza (figura 8.5.2).

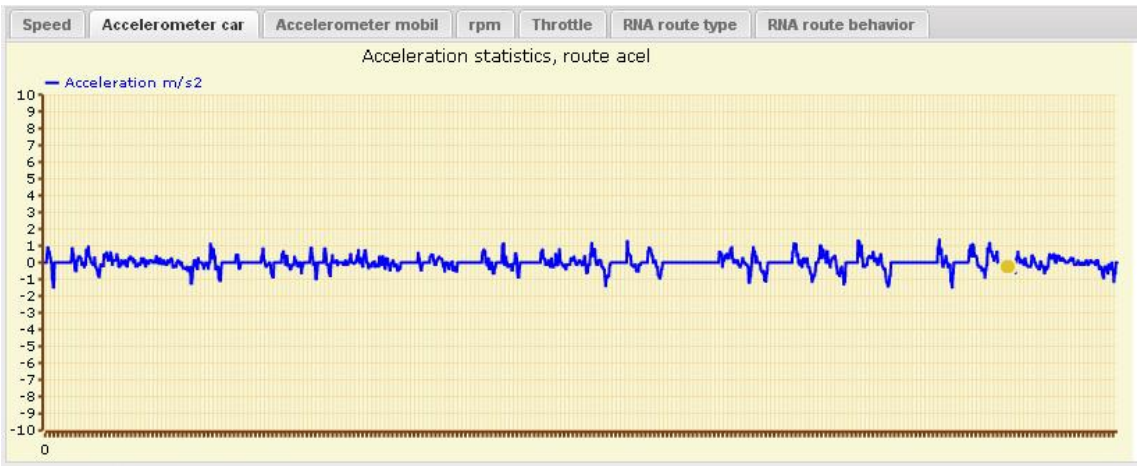


Figura 8.5.2: Gráfica aceleración del vehículo.

3. Revoluciones por minuto: Obtenemos la media y la varianza (Figura 8.5.3).

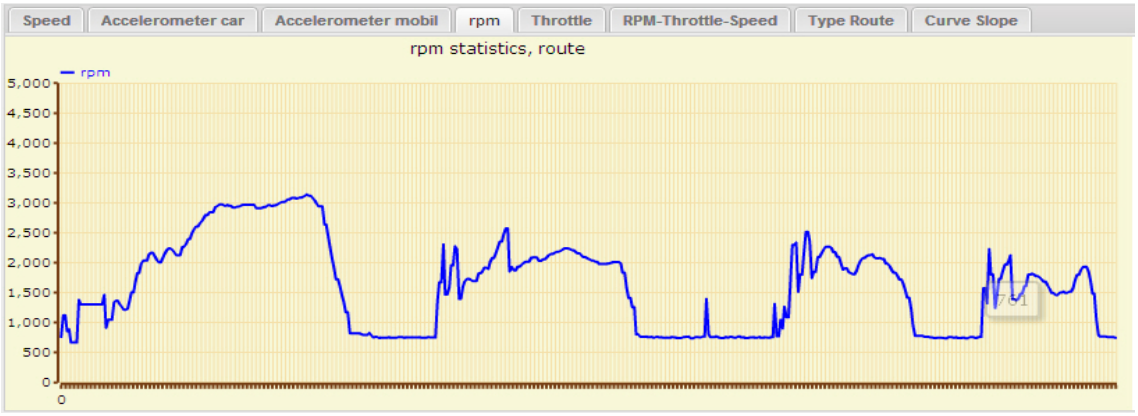


Figura 8.5.3: Gráfica revoluciones por minuto del motor.

La Figura 8.5.4 muestra la salida de la red neuronal que analiza el tipo de vía.

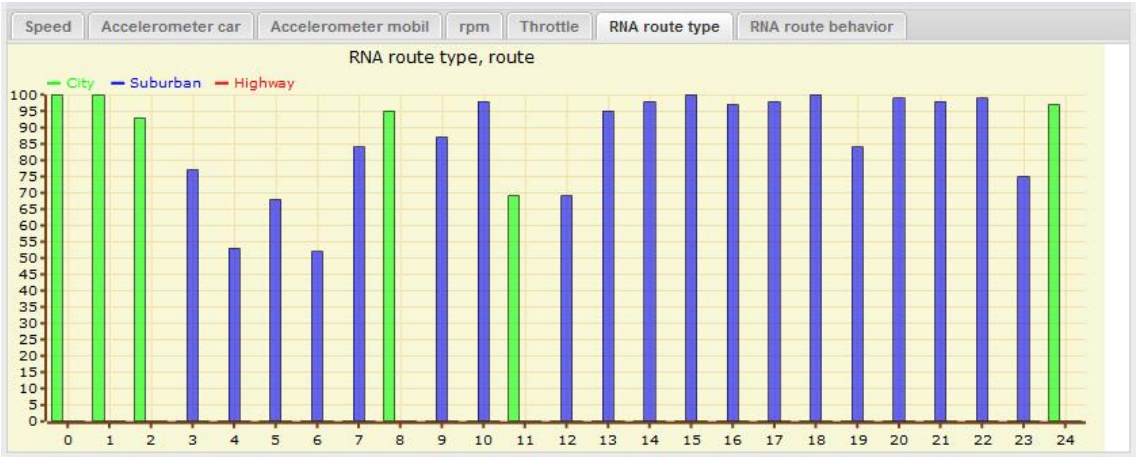


Figura 8.5.4: Respuesta de la Red Neuronal tipo de vía.

La figura 8.5.5 ilustra la salida de la red neuronal que analiza el estilo de conducción.

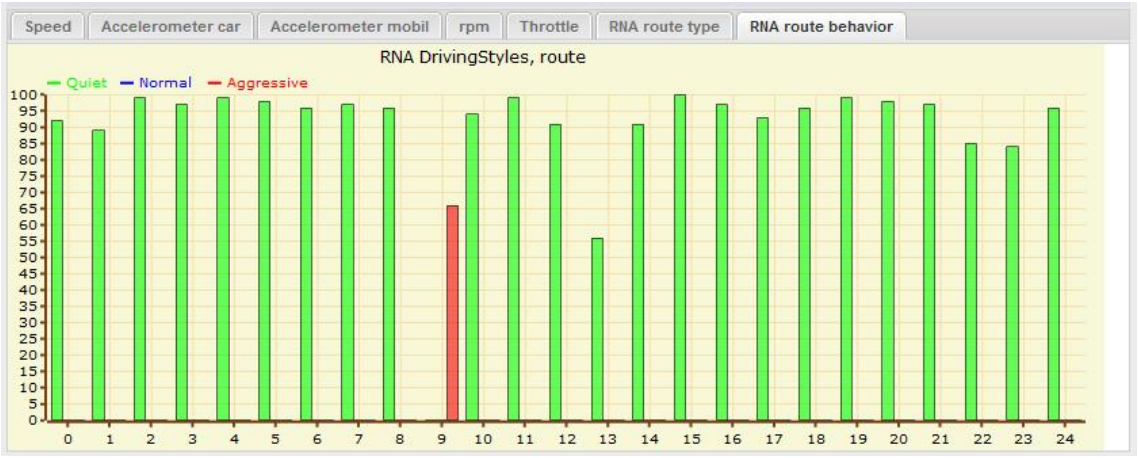


Figura 8.5.5: Respuesta de la Red Neuronal estilo conducción.

También obtenemos el comportamiento de cada usuario ante el volante, analizando el estilo de conducción a partir de todas las rutas enviadas al sistema por el usuario, en la figura 8.5.6 podemos observar la comparativa entre tres sujetos observando el distinto comportamiento entre ellos.

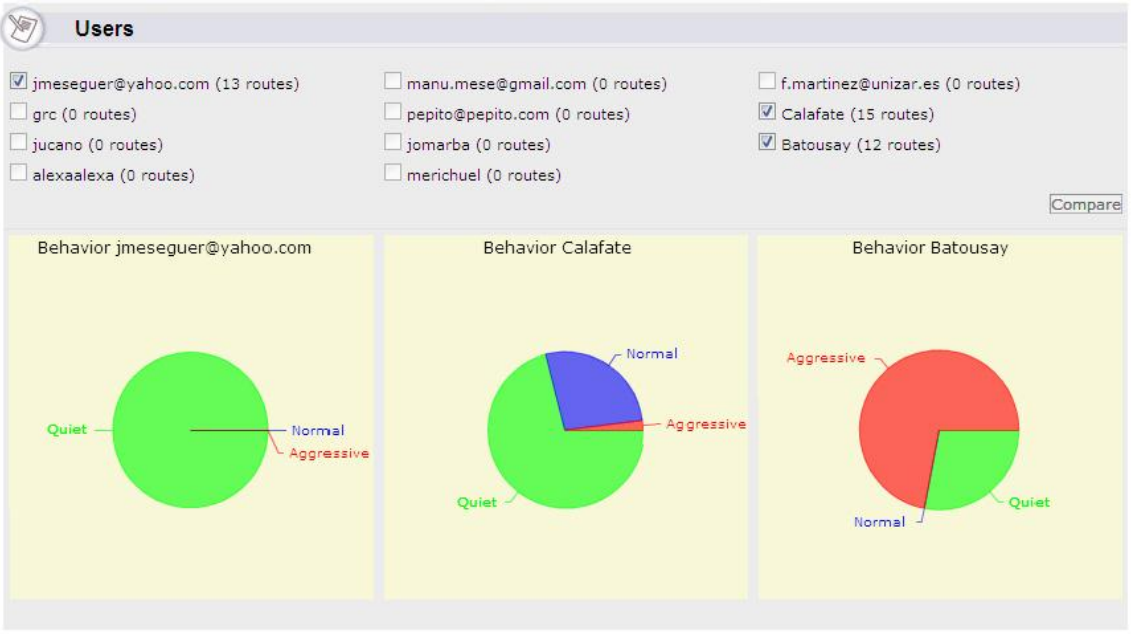


Figura8.5.6: Estudio comparativo del comportamiento de tres usuarios de sistema.

9 CONCLUSIONES Y TRABAJO FUTURO.

Este trabajo fin de Máster viene a demostrar que es posible crear una plataforma que integre dispositivos móviles, datos obtenidos de la *Electronic Control Unit* (ECU) del motor y servicios web para, en base a la aplicación de técnicas de Inteligencia Artificial (IA) como son las redes neuronales, obtener información relativas al estilo de conducción de los usuarios. En nuestro sistema hubo necesidad, además, de entrenar una red neuronal para determinar el tipo de vía por el que circula el conductor, y otra para determinar su estilo de conducción.

Una de las partes más importantes dentro del área de la minería de datos ha sido el generar el conjunto de pruebas y validación para entrenar la red neuronal, ya que este subconjunto es obtenido de forma visual, determinando nosotros si el usuario circula por ciudad o por vías interurbanas. Ha sido también bastante complejo determinar si el conductor tiene una manera de conducir tranquila, normal o agresiva, es decir, como definir los umbrales que separan un estilo de conductor de otro.

Como trabajo futuro seria realmente muy interesante implementar distintas técnicas de Ingeniería Artificial (IA) como sistemas expertos, redes bayesianas o inteligencia artificial basada en comportamientos y realizar un estudio comparativo sobre la eficacia de estas técnicas en la obtención de soluciones para nuestro problema.

Como posibles mejoras en la aplicación estaría la implementación del componente de reconocimiento de vía y del estilo de conducción en el dispositivo móvil, sin necesidad de enviar esos datos recogidos al servidor, ya que computacionalmente no tendríamos ningún problema con los dispositivos móviles actuales.

10 BIBLIOGRAFÍA.

[1] International Organization for Standardization, “ISO 9141-2:1994/Amd 1:1996”,1196.

[2] International Organization for Standardization, “ISO 14230-1:1999: Road vehicles, Diagnostic systems, Keyword Protocol 2000”, 1999.

[3] www.android.com

[4] www.wikipedia.com

[5] Android User´s Guide 2.3 December 13, 2010 AUG-2.3-103 Android™ mobile technology platform 2.3

[6] Android 2.3 Application Development Cookbook Kyle Merrifield Mew ISBN 978-1-849512-94-7

[7] MeeGo 1.0 Mobile Application Development Cookbook Kyle Merrifield Mew ISBN 978-1-849512-94-7

[8] ELM327DS. OBD to RS232 Interpreter. Elm Electronics – Circuits for the Hobbyist.

[9] http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/

[10]http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/manual/JavaNNS-manual.html

[11] JavaNNS, Java Neural Network Simulator, User Manual, Version 1.1 Igor Fischer, Fabian Hennecke, Christian Bannes, Andreas Zell

[12] http://www-ra.informatik.uni-tuebingen.de/SNNS/

[13] Haykin, S. (1994), Neural Networks: A Comprehen-sive Foundation, NY: Macmillan, p. 2

[14] https://play.google.com/store/apps/details?id=org.prowl.torque

[15] https://play.google.com/store/apps/details?id=Scantech.CarGaugeLite

[16] http://code.google.com/p/swfobject/

[17] http://www.ra.cs.uni-tuebingen.de/

[18] Theory of the backpropagation neural network - Neural Networks, 1989. IJCNN., International Joint Conference Hecht-Nielsen, R.

[19] Efficient Classification for Multiclass Problems Using Modular Neural Networks Rangachari Anand, Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka, Member, IEEE

[20] Sistemas Basados en el Conocimiento II: Introducción a la Neurocomputación, Universidad Nacional de Educación a Distancia 2004

11 Anexos.

11.1 Obtener una *API* KEY para google maps.

Para poder mostrar el mapa de la ruta es necesario el uso de la clase *MapView*, tal y como se explica en el apartado 3.1.5 Geolocalización y Mapas, para poder integrar *Google Maps* en la aplicación y mostrar mapas ya que proporciona un *wrapper* con su *API*. Para poder acceder a dicha *API* es necesario registrarse en el servicio de *Google Maps* y aceptar los Términos de Servicio antes de que la clase *MapView* pueda mostrar los mapas. El registro es simple, gratuito y consta de dos partes:

1. Registrar la firma MD5 en el certificado que se usará para firmar la aplicación. Entonces, el servicio de registro de Mapas proporcionará una clave para la *API* que estará asociada al certificado de la aplicación.
2. Añadir la referencia a la clave en cada *MapView* (tanto en los archivos *XML* como directamente en el código).

Si la aplicación se firma en modo *debug* (el SDK utiliza automáticamente un certificado de *debug*). Para generar una firma MD5 con el certificado de *debug*, primero hay que localizar el *debug keystore*.

Una vez localizada la *keystore* se usa la herramienta *keytool* para obtener la firma MD5 ver figura 10.1.1.

```
$ keytool -list -alias androiddebugkey \ -keystore <path_to_debug_keystore>.keystore
\ -storepass android -keypass android
```

Figura 11.1.1: Obtención de la firma MD5.

Una vez obtenida, se accede a la página *http://code.google.com/android/maps-API-signup.html* y se siguen los siguientes pasos:

1. Se introduce la cuenta de *Google.*
2. Se aceptan los Términos de Servicio de la *API* de *Android Maps*.
3. Se pega la firma MD5 en la casilla correspondiente.
4. Se clica en —Generar *API Key.*

Una vez obtenida la clave, hay que añadirla a la aplicación. En el caso de los ficheros *XML* se añade como atributo en los elementos <MapView> tal y como muestra la figura 10.1.2.

```
<com.google.android.maps.MapView android:layout_width="fill_parent"
android:layout_height="fill_parent" android:enabled="true"
android:clickable="true" android:APIKey="APIKey" />
```

Figura 11.1.2: Inclusión de la API Key en el fichero XML de la aplicación.

Para los objetos MapView declarados en el código directamente, se añade la clave como parámetro en el constructor.

```
mMapView = new MapView(this, " APIKey ");
```

Figura 11.1.3: Inclusión de la API Key directamente en el código.

Por último, hay que añadir la librería externa *com.google.android.maps* en el fichero *AndroidManifest.*

```
<manifest XMLns:android="http://schemas.android.com/apk/res/android"
package="com.example.package.name">
...
<application android:name="MiAplicacion" >
<uses-library android:name="com.google.android.maps" />
...
</application>
```

11.2 Código C. Red neuronal backpropagation

```
/*********************************************************
  Entrenada_6_9_3
  --------------------------------------------------------
  generated at Sat Jul 21 11:07:13 2012
  by snns2c ( Bernward Kett 1995 )
*********************************************************/

#include <math.h>

#define Act_Logistic(sum, bias)  ( (sum+bias<10000.0) ? ( 1.0/(1.0 + exp(-sum-bias) ) ) : 0.0 )
#ifndef NULL
#define NULL (void *)0
#endif

typedef struct UT {
          float act;         /* Activation       */
          float Bias;        /* Bias of the Unit */
          int   NoOfSources; /* Number of predecessor units */
   struct UT   **sources; /* predecessor units */
          float *weights; /* weights from predecessor units */
        } UnitType, *pUnit;

  /* Forward Declaration for all unit types */
  static UnitType Units[19];
  /* Sources definition section */
  static pUnit Sources[] =  {
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6,
Units + 7, Units + 8, Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14,
Units + 15,
Units + 7, Units + 8, Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14,
Units + 15,
Units + 7, Units + 8, Units + 9, Units + 10, Units + 11, Units + 12, Units + 13, Units + 14,
Units + 15,

  };

  /* Weigths definition section */
  static float Weights[] =  {
-2.487590, -11.203620, 0.493210, -0.651780, 0.717890, 0.421000,
-4.215170, -19.916691, -0.817190, -0.271100, -0.705370, -0.668200,
-19.610319, -16.347200, -0.990660, -0.982180, -0.244240, 0.063330,
-4.688180, -0.002910, -0.667530, 0.326090, -0.098420, -0.295750,
-5.875830, 10.077310, 0.605210, 0.039770, -0.396100, 0.751950,
24.598810, 0.934050, 0.078710, -0.715320, -0.075840, -0.529340,
-6.104510, 10.445370, 0.687310, 0.993590, 0.999390, 0.223000,
-14.645990, 24.846519, 0.680290, -0.952510, -0.248270, -0.814750,
-24.347130, -1.269100, 0.837580, -0.448230, -0.454210, 0.175820,
-2.570400, -2.698620, -7.687070, 1.196360, 1.101220, -4.121820, 2.037620, 6.153620, 11.725970,
-0.272070, -0.271060, -0.137090, -2.851990, -6.276800, -7.374800, -6.885080, 1.304190, -
10.656210,
5.754560, 11.985340, 10.764420, 0.086450, -5.533430, 11.119040, -5.427600, -14.975080, -
7.891610,

  };

  /* unit definition section (see also UnitType) */
  static UnitType Units[19] =
  {
    { 0.0, 0.0, 0, NULL , NULL },
    { /* unit 1 (noName) */
      0.0, -0.997500, 0,
```

```
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 2 (noName) */
      0.0, , 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 3 (noName) */
      0.0, , 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 4 (noName) */
      0.0, , 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 5 (noName) */
      0.0, , 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 6 (noName) */
      0.0, , 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 7 (noName) */
      0.0, , 6,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 8 (noName) */
      0.0, , 6,
       &Sources[6] ,
       &Weights[6] ,
      },
    { /* unit 9 (noName) */
      0.0, , 6,
       &Sources[12] ,
       &Weights[12] ,
      },
    { /* unit 10 (noName) */
      0.0, , 6,
       &Sources[18] ,
       &Weights[18] ,
      },
    { /* unit 11 (noName) */
      0.0, , 6,
       &Sources[24] ,
       &Weights[24] ,
      },
    { /* unit 12 (noName) */
      0.0, , 6,
       &Sources[30] ,
       &Weights[30] ,
      },
    { /* unit 13 (noName) */
      0.0, , 6,
       &Sources[36] ,
       &Weights[36] ,
      },
    { /* unit 14 (noName) */
      0.0,, 6,
       &Sources[42] ,
       &Weights[42] ,
      },
    { /* unit 15 (noName) */
      0.0, , 6,
       &Sources[48] ,
       &Weights[48] ,
      },
    { /* unit 16 (noName) */
```

```
      0.0, , 9,
       &Sources[54] ,
       &Weights[54] ,
      },
    { /* unit 17 (noName) */
      0.0, , 9,
       &Sources[63] ,
       &Weights[63] ,
      },
    { /* unit 18 (noName) */
      0.0, , 9,
       &Sources[72] ,
       &Weights[72] ,
      }

  };



int D__OBD2_tesis_android_JavaNNS_ruta_solo_parametro_veloci(float *in, float *out, int init)
{
  int member, source;
  float sum;
  enum{OK, Error, Not_Valid};
  pUnit unit;


  /* layer definition section (names & member units) */

  static pUnit Input[6] = {Units + 1, Units + 2, Units + 3, Units + 4, Units + 5, Units + 6};
/* members */

  static pUnit Hidden1[9] = {Units + 7, Units + 8, Units + 9, Units + 10, Units + 11, Units +
12, Units + 13, Units + 14, Units + 15}; /* members */

  static pUnit Output1[3] = {Units + 16, Units + 17, Units + 18}; /* members */

  static int Output[3] = {16, 17, 18};

  for(member = 0; member < 6; member++) {
    Input[member]->act = in[member];
  }

  for (member = 0; member < 9; member++) {
    unit = Hidden1[member];
    sum = 0.0;
    for (source = 0; source < unit->NoOfSources; source++) {
      sum += unit->sources[source]->act
             * unit->weights[source];
    }
    unit->act = Act_Logistic(sum, unit->Bias);
  };

  for (member = 0; member < 3; member++) {
    unit = Output1[member];
    sum = 0.0;
    for (source = 0; source < unit->NoOfSources; source++) {
      sum += unit->sources[source]->act
             * unit->weights[source];
    }
    unit->act = Act_Logistic(sum, unit->Bias);
  };

  for(member = 0; member < 3; member++) {
    out[member] = Units[Output[member]].act;
  }

  return(OK);
}
```