UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

ESTUDIO DE UN CLASIFICADOR DE EVENTOS ACÚSTICOS SOBRE RASPBERRY

Daniel Sanz Montrull

Tutora: **María de Diego Antón**
Cotutora: **María Gema Piñero Sipán**

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

València, 25 de noviembre de 2019

Resum

Aquest TFG es centra en analitzar les possibilitats que ofereix una tarjeta de baix cost com és la Raspberry Pi, per a realitzat la gravació i classificació d'events acústics gravats en entorns *outdoor*. Es durà a terme l'obtenció d'una base de dades d'events acústics gravats de dues maneres: emprant la Raspberry Pi, equipada amb un micròfon, i simultàniament empreant una gravadora d'alta qualitat. Posteriorment, sobre el *software* MATLAB, es durà a terme la classificació dels sons obtinguts. Aquest mètode permetrà evaluar les prestacions que ofereix aquesta tarjeta.

Paraules clau: Classificador de sons, Processat de senyal, Coeficients MFCC, Raspberry Pi platform

Resumen

Este TFG se centra en analizar las posibilidades que ofrece una tarjeta de bajo coste como es la Raspberry Pi, para realizar la grabación y clasificación de eventos acústicos grabados en entornos *outdoor*. Se llevará a cabo la obtención de una base de datos de eventos acústicos grabados de dos formas: empleando la Raspberry Pi, equipada con un micrófono, y simultáneamente empleando una grabadora de alta calidad. Posteriormente, sobre el *software* MATLAB, se llevará a cabo la clasificación de los sonidos obtenidos. Este método permitirá evaluar las prestaciones que ofrece esta tarjeta.

Palabras clave: Clasificador de sonidos, Procesado de señal, Coeficientes MFCC, Raspberry pi platform

Abstract

This TFG is focused on the analysis of the probabilities that a low-cost board, like Raspberry Pi, offers in order to do the recording and clasification of acoustic events that are recorded in outdoor environments. The obtaiment of a data base of recorded acoustic events will be taken place in two ways: using the Raspberry Pi equipped with a microphone and, at the same time, a high quality recorder. Then, on the software MATLAB, the sounds that were obtained previously will be classified. This method will make able the evaluation of the card's performance.

Keywords: Sound classifier, Signal processing, MFCC coefficients, Raspberry pi platform

Índice general

Índice general v
Índice de figuras vii
Índice de tablas vii

1 Introducción 1
1.1 Motivación . . . 1
1.2 Objetivos . . . 1
1.3 Marco teórico . . . 2
1.3.1 *Audio features* . . . 2
1.3.2 *Machine Learning* . . . 3
2 Creación de la base de datos 5
2.1 Trabajo previo . . . 5
2.2 Material . . . 7
2.3 Grabaciones . . . 9
2.4 Segmentación y etiquetado . . . 12
2.4.1 Trabajando con *Reaper* . . . 12
2.4.2 Metodología . . . 13
2.5 Organización de la base de datos . . . 14
3 Elección de *features* 17
3.1 *Features* temporales . . . 18
3.1.1 *Zero Crossing Rate* . . . 19
3.1.2 Índice de modulación AM . . . 20
3.1.3 *Log-Attack Time* . . . 20
3.1.4 *Pitch* . . . 20
3.2 *Features* espectrales . . . 21
3.2.1 *Slope* . . . 21
3.2.2 *Decrease* . . . 21
3.2.3 *Flux* . . . 21
3.2.4 *Roll-Off* . . . 22
3.2.5 *Centroid* . . . 22
3.2.6 *Spread* . . . 22
3.2.7 *Skewness* . . . 23
3.2.8 *Kurtosis* . . . 23
3.3 MFCC . . . 23
3.4 Extracción de *features* . . . 24
4 Clasificadores 27
4.1 Tipos de clasificadores . . . 27
4.1.1 *Random Forest* . . . 27
4.1.2 *Support Vector Machine* . . . 28
4.2 Clasificación multiclase . . . 28
4.3 Validación cruzada . . . 29
4.4 Implementación . . . 31

5 Resultados **33**
5.1 Evaluación con las muestras tomadas con la grabadora 34
5.1.1 Selección de características 35
5.2 Evaluación por correlación cruzada de bases de datos 37
6 Conclusiones **45**
7 Líneas futuras **47**
Bibliografía **49**

Anexos
A Códigos utilizados **51**
A.1 Código en Python utilizado para grabar con la Raspberry y el micrófono USB 51
B *Datasheet* del micrófono *UMIK-1*. **53**

Índice de figuras

2.1 Ubicaciones de las grabaciones . . . 6
2.2 Micrófono MiniDSP UMIK-1 . . . 7
2.3 Grabadora de mano H4n Pro . . . 8
2.4 Montaje de la Raspberry con la batería y el micrófono USB . . . 9
2.5 Capturas del cliente SSH *PuTTY* . . . 10
2.6 Disposición del equipo de grabación en la *Ubicación 1* . . . 11
2.7 *Ubicación 1* . . . 11
2.8 Grabaciones en la *Ubicación 2* . . . 12
2.9 Interfaz de usuario de *Reaper* . . . 13
2.10 Metodología de clasificación . . . 14
2.11 Organización de los directorios de la base de datos de ruidos . . . 16

3.1 *Mid-term processing* . . . 18
3.2 Extracción de las *features* espectrales y cepstrales . . . 19
3.3 Descripción del *script* que guarda los vectores de características . . . 25
3.4 Descripción de la función que extrae las *features* . . . 26

4.1 Funcionamiento de las técnicas OVA y OVO . . . 29
4.2 *K-fold cross-validation* . . . 30
4.3 *Leave-one-out cross-validation* . . . 30
4.4 Validación cruzada aleatoria . . . 31
4.5 Descripción de la función que realiza las pruebas con el clasificador . . . 32

5.1 Resultados tras la primera prueba . . . 40
5.2 Número de muestras limitado a un máximo de 200 . . . 41
5.3 Resultados de precisión total tras limitar las muestras . . . 41
5.4 Resultados para 7 clases . . . 42
5.5 Ejemplo de pares de valores de *features* . . . 42
5.6 Precisión total de cada clasificador según las *features* utilizadas . . . 43
5.7 Resultados de precisión por clases . . . 43
5.8 Precisión total con `161median` y `161minmax` . . . 44
5.9 Matriz de confusión utilizando `161median` con un *Random Forest* . . . 44

Índice de tablas

1.1 Organización de un *dataset* . . . 4

2.1 Extracto de las especificaciones de la grabadora de mano . . . 8

2.2 Especificaciones técnicas de la Raspberry Pi 3 Modelo B 9
2.3 Fragmentos de audio de cada clase de ruido 15

3.1 Duraciones de las tramas en muestras y milisegundos para una frecuencia de muestreo de 48 kHz 19
3.2 Features extraídas para cada clase 25

5.1 Todas las combinaciones de *features* utilizadas 36
5.2 Resultados de precisión con distintos *datasets* 38

CAPÍTULO 1
Introducción

En este capítulo se pretende exponer la motivación que lleva a realizar este trabajo, los objetivos que tiene e introducir los conceptos teóricos necesarios para una comprensión de la memoria. Cabe destacar que los conceptos teóricos que se introducen serán desarrollados más tarde en sus respectivos capítulos.

1.1 Motivación

Debido al auge de la concentración de la población en las grandes ciudades surge la necesidad de gestionar los espacios urbanos de una manera más eficiente. Gracias al progreso en el ámbito del Internet de las Cosas, se pueden encontrar herramientas que faciliten esta labor. Un clasificador de eventos acústicos sobre Raspberry es una herramienta que puede facilitar la gestión en espacios urbanos, ya que, sólo colocando un dispositivo de un tamaño pequeño y con un coste mínimo, permite conocer en todo momento el nivel de ruido y la fuente de ruido que lo genera. Esto también podría permitir identificar qué fuentes de ruido tienen más importancia en cada zona de la ciudad.

Es bien sabido que la contaminación acústica y la exposición a grandes niveles de sonido causa problemas en la salud, desde pérdidas de auditivas hasta problemas de sueño. Un conocimiento de qué fuentes de ruido son las causantes de estos problemas de salud en cada persona podría ser de gran ayuda para que las distintas organizaciones puedan adoptar medidas que sirviesen para prevenir estas patologías y mejorar el bienestar de la población.

Por otra parte, la implementación de un clasificador a tiempo real sobre un dispositivo de un tamaño reducido, podría ser una solución que permita ayudar a personas con discapacidad auditiva a reconocer los sonidos de su entorno, como podría una señal acústica procedente de un cláxon o de un agente de la policía. Esto les podría ayudar a obtener más información sobre el entorno.

1.2 Objetivos

Este trabajo tiene como objetivo cuantificar cómo la calidad del material utilizado en la creación de una base de datos de ruidos de ciudad influye en la calidad de un sistema implementado con calidad profesional capaz de clasificar estas fuentes sonoras. Concretamente se centra en determinar cómo influyen las características de un micrófono a la hora de crear una base de datos y a la hora de clasificar con algoritmos de *Machine Learning*. Determinar esta influencia permite ahorrar costes en material a la hora de utilizar

una red de sensores de ruidos urbanos o a la hora de desarrollar un dispositivo portátil con bajo coste que permita realizar tareas de clasificación.

Por otro lado, se estudiará el proceso completo y los pasos que se llevan a cabo a la hora de solucionar un problema con *Machine Learning*.

1. Diseño de una base de datos con sonidos urbanos.
2. Elección de los escenarios de grabación.
3. Captación de las muestras que conformarán el conjunto de sonidos urbanos.
4. Segmentación y etiquetado de las grabaciones realizadas.
5. Extracción y selección de las características que se utilizarán para distinguir entre las distintas clases de sonidos.
6. Entrenamiento de un algoritmo de *Machine Learning*.
7. Optimización del sistema.

Al realizar la optimización del sistema, conocer cada parte de este proceso es determinante para decidir de qué manera se realizará esta optimización. Cuando se realiza un análisis de los resultados obtenidos, comprender por qué se están obteniendo unos determinados resultados ayuda en la toma de decisiones para llevar a cabo una optimización. Unas decisiones llevan unos costes temporales, económicos y computacionales distintos con respecto a otras [1].

1.3 Marco teórico

1.3.1. *Audio features*

Las *features* o características son propiedades que aportan información descriptiva de un determinado fenómeno. Extraer estas características del audio es esencial en la parte de procesado que se realiza en los algoritmos de *Machine Learning*. El proceso de extracción de estas *features* se podría ver como un proceso de reducción de datos, ya que, se desea trabajar con el menor número posible de características para que la tarea de procesado se realice de una forma más rápida, además de simplificar la señal de audio. La señal original es complicada de procesar directamente por su tamaño. Por lo tanto, es necesario transformar los datos del audio en otros que representen las propiedades de las señales originales reduciendo el volumen de datos que ocupa una grabación. Dependiendo de para qué aplicación se vaya a realizar este procesado que se ha descrito, siendo en el caso presente para clasificación de eventos acústicos, es importante saber que *features* van a ser más adecuadas de procesar. [2]

Tal y como se explica en [3], se puede dividir una grabación de audio en 4 niveles siguiendo una jerarquía:

1. **Tramas:** Fragmentos de audio con una duración de 20 a 30 ms. Son la unidad más pequeña en procesado de audio.
2. **Segmentos:** Un segmento está compuesto de un número de tramas. La duración de estos segmentos viene fijada en una duración máxima.
3. **Toma:** Se define como toma una unidad estructural que contiene la misma clase de ruido. Puede estar formada de uno o más segmentos.

4. **Unidad semántica de alto nivel:** Estructura de audio con un contenido semántico rico, que está formada por diferentes combinaciones de tomas. Esta unidad pertenecería, en este caso, a una grabación.

En este trabajo la segmentación de las grabaciones enteras se renderizarán como tomas, guardando estas tomas como elementos que conforman la base de datos. La división de las tomas por segmentos se realizará en MATLAB antes de iniciar la tarea de extracción de las características del audio. Por último, la extracción de *features* se realiza sobre un conjunto de tramas que forman un mismo segmento.

Basándose en los resultados de [4], se elige una duración de segmento máxima de 4 segundos, ya que, por debajo de este valor, la precisión del clasificador empieza a descender de forma notable. En cambio, las diferencias en el rendimiento del clasificador estableciendo estas duraciones de segmento a 6 segundos no aporta una diferencia notable con respecto a utilizar una duración máxima de segmento de 4 segundos.

En el Capítulo 3, *Elección de features*, se explicarán las *features* que se utilizarán de una forma más detallada.

1.3.2. *Machine Learning*

El *Machine Learning* se encarga de que un ordenador pueda aprender a realizar una tarea específica. Algunas de las aplicaciones que se pueden encontrar en el *Machine Learning* son entre otras:

- Reconocer caracteres en una foto.
- Ayudar en el diagnóstico de personas con enfermedades graves mediante reconocimiento de imágenes médicas.
- Separar material según su calidad.
- Clasificar diferentes tipos de vino.
- etc.

Es importante, antes de realizar una tarea de clasificación, saber qué características del objeto de dicha clasificación son relevantes para distinguirlo de otros objetos. [5]
Por ejemplo, en caso de que se desee predecir el precio de una vivienda, es posible que el tamaño de dicha vivienda sea una característica importante, en cambio, el número de sillas que tiene la vivienda, no parece ser un dato a tener en cuenta si se quiere hacer una predicción. [1]

Dentro del *Machine Learning* se distinguen entre dos áreas: aprendizaje supervisado y no supervisado.
En el aprendizaje supervisado, se tienen las entradas (muestras de la base de datos) etiquetadas con las salidas que se deberían de obtener. Las salidas de las muestras etiquetadas pueden ser un valor continuo (en el caso de tener un problema de regresión) o un valor discreto que represente una clase (en el caso de los problemas de clasificación) [1].
En el caso que se ocupa, se tiene un problema de clasificación en el que las muestras irán etiquetadas en una clase que representará la fuente sonora predominante en la toma.
Por otra parte, el aprendizaje no supervisado se asocia a construir modelos para analizar los datos y encontrar patrones y similitudes entre una cantidad de datos de entrada. [5, 1].

El trabajo del algoritmo de *Machine Learning* es converger al mejor clasificador posible de tal manera que clasifique con el mínimo error posible las muestras que se utilizan para

entrenarlo. El objetivo final es que este clasificador sea capaz de etiquetar correctamente muestras que no se han utilizado en el entrenamiento del algoritmo.

Las muestras que se utilizan tanto para entrenar como para obtener la precisión del clasificador, están ordenadas en un *dataset* (conjunto de datos) de m muestras y n *features*, formando una tabla (o matriz) donde la fila indica el número de muestra y la columna indica la *feature* [1]. Por lo tanto, en una celda de la matriz que forma el *dataset* se tiene el valor de de una determinada *feature* de una determinada muestra. La clase a la que corresponde cada muestra, se puede indicar o bien por el nombre de carpeta en el que está guardada dicha muestra (como se ha hecho en este caso), con una tabla donde se indica para cada muestra su valor de salida (o clase a la que pertenece) o etiquetando el archivo que contiene la muestra. En la Tabla 1.1 se muestra la organización de dicho *dataset*, además de dar una idea visual del funcionamiento de esta estructura. En este trabajo, para indicar el valor de salida que debería de tener cada muestra, se genera un vector de m valores de salida con MATLAB utilizando el nombre del directorio en el que están guardadas las muestras como salida. Siendo el valor i-ésimo del vector de salidas correspondiente a la muestra i-ésima del *dataset*.

Tabla 1.1: Organización de un *dataset*

Número de muestra	**Matriz de entradas**				**Vector de salidas (clases)**
Muestra 1	*feature* 1	*feature* 2	...	*feature n*	Clase a
Muestra 2	*feature* 1	*feature* 2	...	*feature n*	Clase b
Muestra 3	*feature* 1	*feature* 2	...	*feature n*	Clase c
...	...	...	...	...	...
Muestra m	*feature* 1	*feature* 2	...	*feature n*	Clase b

En el Capítulo 4, *Clasificadores*, se encuentran más detalles sobre cómo se han implementado estos clasificadores y cómo funcionan.

Este trabajo se encuentra situado dentro del área de aprendizaje supervisado, donde se tienen unas muestras (tomas grabadas), a las que han extraído unos datos representativos (*features*) y además se conoce en qué tipo de ruido deberían de ser catalogadas por parte del clasificador. El objetivo, por lo tanto, es encontrar un clasificador que sea capaz de catalogar correctamente una muestra de ruido que no haya observado con anterioridad.

CAPÍTULO 2
Creación de la base de datos

Para cumplir con el objetivo de este trabajo es necesario elaborar una base de datos propia con grabaciones tomadas desde dos dispositivos de grabación *hardware*, que además, incorporen micrófonos diferentes para comprobar si la calidad del micrófono desde el que se toman las grabaciones influye tanto en la creación de un *dataset* y/o en la precisión que podría tener un dispositivo con un clasificador implementado.

2.1 Trabajo previo

Como punto de partida se ha establece como objetivo que el clasificador que se va a entrenar sea capaz de distinguir entre 4 clases de sonido como mínimo. Teniendo esto en cuenta, se empieza decidiendo que tenga que distinguir entre las siguientes 4 fuentes de ruido: Tráfico, sirenas, voces y desconocido. Estas fuentes de ruido se denominan **clases**. La clase *desconocido* se utiliza para clasificar aquellos eventos acústicos que no se pueden catalogar en ninguna de las clases anteriores.
Según las diferentes fuentes sonoras que se encuentren en las grabaciones o según los resultados que se obtengan en la tarea de clasificación, estas clases que se han establecido inicialmente podrán variar.

Cuando se ha decidido qué clases se van distinguir, es necesario elegir las ubicaciones en las que se encontrarán las fuentes de ruido pertenecientes a dichas clases. Se establece como requerimiento mínimo disponer de una hora de grabación por ubicación, ya que, es necesario que la base de datos contenga una cantidad de muestras lo suficientemente grande para que el clasificador sea capaz de aprender las características que tienen los sonidos pertenecientes a una clase para que después pueda generalizar correctamente un caso que no ha observado anteriormente.

Para la creación de una base de datos propia con sonidos de ciudad, se han elegido dos ubicaciones: Enfrente del *Hospital Clínic Universitari de València* (en las coordenadas 39.478361, -0.361861) (a partir de ahora se pasará a llamar *Ubicación 1*) y la otra en los jardines de Rectorado de la UPV mientras se realizaban las actividades de *l'Escola d'Estiu* (en las coordenadas 39.481028, 0.346278) (a partir de ahora se pasará a llamar *Ubicación 2*).

En la Figura 2.1 se observan dos capturas realizadas con *Google Maps* donde están marcados los puntos en los que se realizarán las grabaciones de muestras para la base de datos.

Se consideró la elección del parque Gulliver de Valencia como zona de grabación para captar ruidos de voces, pero se descartó porque el día en que se fue a grabar había mú-

Figura 2.1: Ubicaciones de las grabaciones. En la imagen de arriba se encuentra la *Ubicación 1* y en la de debajo la *Ubicación 2*. Fuente: *Google Maps* [6]

sica de fondo. Esto habría perjudicado considerablemente a las muestras de voz, siendo necesario que estén lo más limpias posibles de ruidos de fondo o interferentes.

La motivación para elegir la *Ubicación 1* fue la posibilidad de poder captar principalmente sonido de tráfico y sonidos de sirenas en esta zona. La ubicación elegida permite obtener estos dos tipos de ruido, ya que, la Avenida de Blasco Ibáñez de Valencia es una vía con bastante tráfico y al situarse enfrente de un hospital, existe una mayor probabilidad para obtener sonidos de sirenas de ambulancias. Por otra parte, lo que motivó a la elección de la *Ubicación 2* fue la posibilidad de disponer ruidos de voces tanto de niños como de adultos a casusa de las actividades que se realizan en verano en la UPV, además, de que esta ubicación permite obtener muestras de voces menos contaminadas por el sonido de tráfico de la ciudad, a diferencia de las muestras de voces que se han obtenido en la *Ubicación 1* que disponen de más ruido de tráfico de fondo.
Como después se observará en la Sección 2.4, se encontrarán en cada una de las dos ubicaciones, más fuentes de ruido a tener en cuenta.

2.2 Material

El material necesario para realizar las grabaciones que conformarán la base de datos está compuesto por: una grabadora de mano (se llamará también *Micrófono 1*), un micrófono USB (al que también se llamará *Micrófono 2*), una Raspberry Pi, una batería portátil para la Raspberry, un cable de Ethernet y un ordenador portátil.

Por otra parte, el material que se ha utilizado para la segmentación y etiquetado del audio grabado, es una interfaz de audio externa USB, unos auriculares y un ordenador. No sería estrictamente necesario que la calidad del equipo que se utiliza en la parte de procesado de audio sea la mejor, pero si la suficiente como para poder distinguir qué fuentes de sonido predomina en cada instante de la grabación. Por ejemplo, no sería conveniente componer las muestras de la base de datos utilizando auriculares que tienen la respuesta en frecuencia muy distorsionada o que sean de mala calidad. Así que, en este caso, es conveniente utilizar un equipo con una respuesta en frecuencia lo suficientemente plana.

El micrófono USB que se utiliza para hacer las grabaciones es un *MiniDSP UMIK-1* con una esponja antivientos y un trípode. La señal procedente de este micrófono se ha muestreado a una frecuencia de muestreo de 48 kHz con una resolución de 16 bits. Se conectará vía USB a la Raspberry y desde el portátil se mandará el comando para realizar la grabación. En la Figura 2.2 se puede observar el micrófono y los accesorios que se utilizan. En el Apéndice B se encuentran las especificaciones técnicas de este micrófono.

Figura 2.2: Micrófono MiniDSP UMIK-1. A la izquierda muestra el micrófono con la pantalla antivientos. A la derecha se tiene el micrófono con el soporte y conectado mediante USB.

La grabadora de mano que se ha utilizado es una *Zoom H4n Pro* [7] con un paravientos que viene incluido. Las especificaciones más relevantes de la grabadora se encuentran detalladas en la Tabla 2.1. Para realizar las grabaciones de campo se han utilizado pilas para la alimentación. La *Zoom H4n Pro* ofrece una calidad de sonido bastante buena, dispone de un micrófono estéreo, además de contar con un gran abanico de configuraciones. Las opciones que se han utilizado para este trabajo son las que permiten cambiar la resolución de los bits, la frecuencia de muestreo y la ganancia de entrada del micrófono (útil para evitar que los ruidos demasiado potentes saturen el micrófono). Otra característica que ha resultado de gran utilidad es la de introducir marcas temporales en la grabación que se está realizando en ese momento (máximo 99 marcas por archivo de grabación). En la Figura 2.3 se observa la grabadora sin la pantalla antivientos y con la pantalla antivientos. El micrófono estéreo está situado en la parte de arriba de la grabadora de mano.

Figura 2.3: A la izquierda se observa la grabadora de mano que se ha utilizado. A la derecha se observa la grabadora con el paravientos.

Tabla 2.1: Extracto de las especificaciones de la grabadora *Zoom H4n Pro*. Extraído de la página web de la grabadora [7]

Micrófono	Micrófono estéreo X/Y unidirecional integrado ajustable entre 90º y 120º
Sensibilidad	-45 dB/1 Pa a 1 kHz
Ganancia de entrada	De -16 dB a +51 dB
Máxima presión sonora de entrada	140 dB SPL
Cuantificación	16 / 24 bits
Frecuencias de muestreo	44.1 / 48 / 96 kHz
Marcadores	Hasta 99 por grabación
Autonomía	Hasta 6 horas con dos pilas AA (más de 11 horas en modo *Stamina*)

En este caso, el micrófono USB no tolera una frecuencia de muestreo de 44.1 kHz y por ello se ha utilizado la frecuencia de muestreo para las grabaciones de 48 kHz en ambas bases de datos. Para clasificación de eventos acústicos, con una resolución de 16 bits es suficiente, ya que, es una resolución que se ha utilizado en trabajos anteriores.

La Raspberry que se ha utilizado es una *Raspberry Pi 3 Model B* utilizando como sistema operativo la distribución Raspbian [8]. En la Tabla 2.2 se muestran las características más relevantes de la Raspberry [9]. La Raspberry se ha configurado incluyendo la librería *pyaudio* [10] de *Python* y activando el protocolo SSH (*Secure Shell*) el cual utiliza el puerto 22.
El protocolo SSH [11] servirá para establecer una conexión en un canal seguro entre la Raspberry y el ordenador, y así, poder ejecutar comandos en la Raspberry de manera remota y descargar las grabaciones al ordenador portátil una vez se hayan realizado. La conexión entre el ordenador y la Raspberry se realizará a través de un cable Ethernet y configurando ambos dispositivos con una dirección IP fija. El ordenador hace de cliente SSH y la Raspberry hace de servidor SSH, para el ordenador se utiliza el cliente SSH *PuTTY* [12] con licencia libre.
Para realizar las grabaciones se ha utilizado un script escrito en *Python* configurado para guardar un archivo en formato .wav cada 10 minutos y el cuál se ejecuta de forma remota desde el ordenador con el cliente SSH descrito anteriormente.
Las grabaciones de la Raspberry se descargan al ordenador utilizando el *software FileZilla*.

Tabla 2.2: Especificaciones técnicas de la Raspberry Pi 3 Modelo B. Extraídas de la página web oficial de Raspberry [9].

Fecha de lanzamiento	Febrero del 2016
Procesador	Quad Core 1.2GHz Broadcom BCM2837 64bit
Memoria RAM	1 GB
Alimentación	5 V / de 2.1 a 2.5 A
Conectividad	4 puertos USB, Ethernet, Wi-Fi
Almacenamiento	Tarjeta MicroSD

FileZilla es un cliente FTP gratuito que, en este caso, utiliza el protocolo SFTP y el puerto 22 para establecer una conexión con la Raspberry, y así, descargar la carpeta en la que se encuentran las grabaciones [13]. Como alimentación, se utiliza una batería portátil con una capacidad de 16750 mAh con una salida de 5V y 2.4A, características que son suficientes para dar alimentación a la Raspberry y al micrófono USB durante el tiempo en el que se realizan las grabaciones. En la Figura 2.4 se muestra como quedarían todos los elementos nombrados en este párrafo conectados a la Raspberry.

Figura 2.4: Montaje de la Raspberry con la batería y el micrófono USB.

Cabe resaltar que todo el material que se ha utilizado en el proceso de obtención de muestras para constituir el *dataset* cabe en una mochila. Esto facilita mucho el proceso de captación de muestras.

2.3 Grabaciones

Cuando ya se tiene todo el material preparado, se dispone a ir a las dos ubicaciones elegidas para realizar las grabaciones que servirán como muestras de las clases de sonidos en la base de datos. Los pasos seguidos en la realización de las grabaciones en cada zona se muestran a continuación:

1. Llegada a la ubicación
2. Conexión y encendido de los dispositivos
3. Con el cliente *PuTTY* se realiza la conexión SSH a la Raspberry (IP: `192.168.1.23`)

4. Inicio de sesión en la Raspberry con las credenciales predeterminadas. Usuario: `pi`. Contraseña: `raspberry`. Con esto se puede utilizar el terminal de la Raspberry.

5. `rm -r wavs`. Borra el directorio `wavs` y el contenido que pudiese tener de grabaciones anteriores

6. `mkdir wavs`. Creación del directorio `wavs`

7. `python3 recording.py`. Ejecución del script de Python que permite grabar desde el micrófono USB

8. Se inicia la grabación con la grabadora

9. Se dan palmas para luego sincronizar las grabaciones de ambos micrófonos. Como la Raspberry crea un archivo en formato wav cada 10 minutos, este paso se repite cada vez que se crea un archivo nuevo en la Raspberry

10. Se insertan marcas temporales en la grabadora cada vez que ocurre algún evento acústico de interés

11. Una vez finalizan las grabaciones en la Raspberry, se deja de grabar con la grabadora

12. Se apagan, se desconectan y se guardan todos los dispositivos utilizados.

En la Figura 2.5 se muestran capturas del cliente SSH *PuTTY*. El *script* llamado `recording.py` utilizado para grabar con el micrófono USB implementado en *Python* se encuentra en Apéndice A.

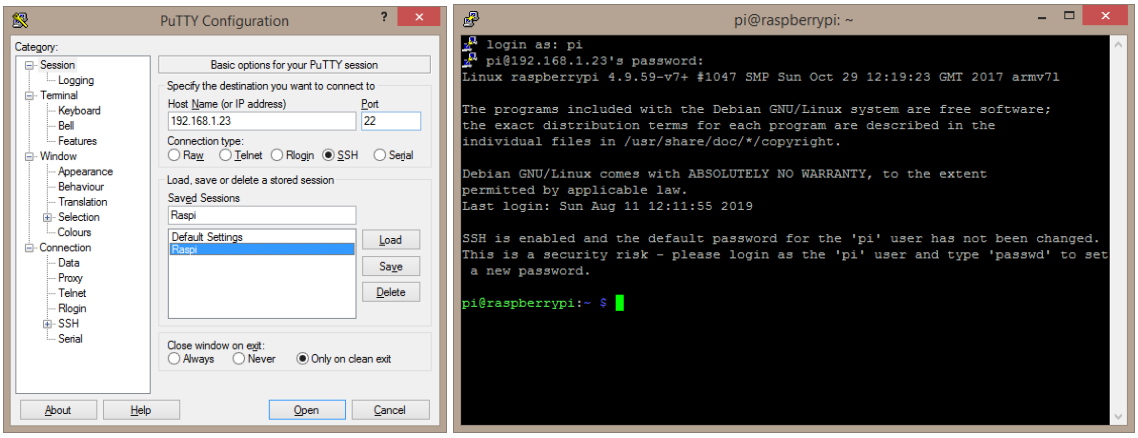


Figura 2.5: Cliente SHH *PuTTY*. En la captura de la izquierda se muestra cómo se realiza la conexión a la Raspberry. En la de la derecha se muestra el cliente una vez establece la conexión y se autentifica con la Raspberry. Fuente: Elaboración propia.

Los días 4 y 5 de julio de 2019 se realizan las grabaciones en la *Ubicación 1* por la tarde, el día 4 sobre las siete de la tarde y el día 5 sobre las siete y cuarto de la tarde. En esta ubicación se halla en una avenida con bastante tráfico y enfrente de un hospital, también circulan peatones cerca de la ubicación en la que se realizan las grabaciones, ya que, el punto se encuentra cerca de un paso de peatones. En esta zona se encuentran ruidos destacables procedentes de tráfico rodado, voces, sirenas, pájaros y palmas (las que se han utilizado para sincronización). Se ha utilizado la pantalla antivientos porque como se ha podido revisar después de una grabación de prueba con la grabadora, el viento generaba un ruido significativo en las muestras. Antes de grabar, se ha guardado todo el equipo en la mochila a excepción del micrófono USB y la grabadora, esto se muestra en

la Figura 2.6, siendo así como se ha dispuesto el material durante las grabaciones de las muestras. También se observa en la Figura 2.7 fotos tomadas desde el punto en el que se han realizado las grabaciones en esta ubicación.

Figura 2.6: Disposición del equipo de grabación en la *Ubicación 1*.

Figura 2.7: *Ubicación 1*.

En la *Ubicación 1* se grabó dos días porque, a pesar de que el ordenador portátil estaba configurado para no entrar en suspensión una vez se cerraba la tapa, no se tuvo en cuenta que después de 30 minutos sin actividad entraba en estado de suspensión. Por lo tanto, del primer día se tienen 30 minutos de grabación, entrando el ordenador en suspensión y perdiendo la conexión con la Raspberry. Esto causó que la Raspberry dejase de grabar teniendo sólo los 30 primeros minutos de grabación. Al día siguiente, se vuelve a una hora similar y ya se consiguen realizar 70 minutos de grabación, tal y como se tenía pensado previamente. Esto se hizo configurando el ordenador para que no entrase en suspensión. Se tiene un total de 100 minutos de grabaciones de esta ubicación aproximadamente.

En la *Ubicación 2* se grabó en torno a la una del mediodía del día 5 de julio de 2019 mientras se desarrollaban en la zona las actividades de la *Escola d'Estiu* de la UPV. Esta ubicación se encuentra en un jardín donde hay niños jugando, monitores gestionando las actividades, ruidos de un silbato, palmas que se utilizan para sincronizar el audio y ruido del tráfico que llega desde la *Avinguda dels Tarongers*. Al igual que en la *Ubicación 1*, también se ha utilizado la pantalla antivientos para las grabaciones en ambos micrófonos. En la Figura 2.8 se muestran fotos realizadas en esta ubicación.

En ambas ubicaciones fue de gran utilidad utilizar las marcas temporales de la grabadora. Para usar las marcas temporales, simplemente hay que pulsar al botón de *grabar*

Figura 2.8: A la izquierda se muestra una foto de la *Escola d'Estiu* en la *Ubicación 2*. A la derecha se muestra la disposición del equipo de grabación en esta zona.

una vez esté grabando. Cada vez que sucedía un evento acústico relevante, se introducía una marca temporal en el archivo wav. En el segundo día de la *Ubicación 1*, la grabación de la grabadora se hizo en dos partes, para evitar llegar así al máximo (99 marcas) y no quedarse sin poder introducir más marcas temporales. Esta funcionalidad resultará de gran utilidad en la parte de la segmentación y el etiquetado, ya que, permite identificar más fácilmente los eventos acústicos relevantes o momentos de la grabación que se deseen resaltar.

2.4 Segmentación y etiquetado

Después de haber realizado todas las grabaciones, se procede a segmentar las grabaciones en tomas y etiquetarlas según la clase de sonido que predomine en el fragmento que se esté tratando en la tarea de segmentado, tal y como se explica en la Subsección 1.3.1.
En la Sección 2.1, *Trabajo previo*, se habían planteado 4 clases de ruido en un principio, pero dada la variedad de fuentes de ruido que se encuentran en las grabaciones, se decide ampliar a 10 clases de ruido, al menos en el etiquetado de audio. En caso de tener una clase con muestras escasas o que dé lugar a confusión en la red neuronal (debido a que sean sonidos muy parecidos), siempre se puede dejar de tener en cuenta la clase conflictiva o introducirla en la clase *desconocido*.

2.4.1. Trabajando con *Reaper*

Para realizar esta tarea de procesado se ha utilizado el software *Reaper* [14]. *Reaper* es un *DAW (Digital Audio Workstation)* que permite procesar las grabaciones utilizando varias pistas, ofrece una manera muy sencilla de segmentar las pistas, facilita el renderizado de las muestras y cuenta con una interfaz fácil de utilizar. En este programa se trabaja en el mismo formato que se encuentran las grabaciones (WAV, 48 kHz, 16-bit, mono).

Para empezar, se crea un nuevo proyecto y se cargan las grabaciones de la Raspberry y las de la grabadora. Luego se encuentran los fragmentos de audio en los que se dan las palmas y se sincronizan las grabaciones de la Raspberry con las de la grabadora. Se añade una tercera pista con las grabaciones de la grabadora simplemente para tener las marcas como referencia, ya que, una vez se renderizan los segmentos se eliminan del proyecto y no es necesario seguir teniendo las marcas temporales de referencia. En la Figura 2.9 se

muestra el interfaz de usuario de *Reaper*, teniendo en la primera pista la grabación de la grabadora, en la segunda pista las grabaciones de la Raspberry y en tercer lugar se tiene la pista con el archivo wav de la grabadora con las marcas temporales que servirá como referencia. La cuarta y quinta pista se utilizan para introducir las tomas de la misma clase de ruido que se vayan haciendo de la pista de la Raspberry y de la grabadora en paralelo. De esta manera, se acaban seleccionando todas las tomas y se renderizarán a una carpeta que indique a qué clase pertenece.
La longitud de la toma viene determinada por la duración de la fuente de ruido predominante desde el inicio hasta el final de esta. Entonces, el cambio del evento acústico es el que determina el final de una toma. En consecuencia, existen tomas de audio con duraciones muy diversas en la base de datos. Si estas tomas duran más de 4 segundos no es problema dado a que la función de MATLAB que extrae las *features* ya se encarga de tratarlas como segmentos independientes, pudiendo así extraer de un archivo de audio más de un vector de *features*.
Hay que destacar que las grabaciones de la grabadora están en estéreo. Al renderizar las tomas a audio *mono*, se genera este único canal haciendo un promediado de ambos canales.

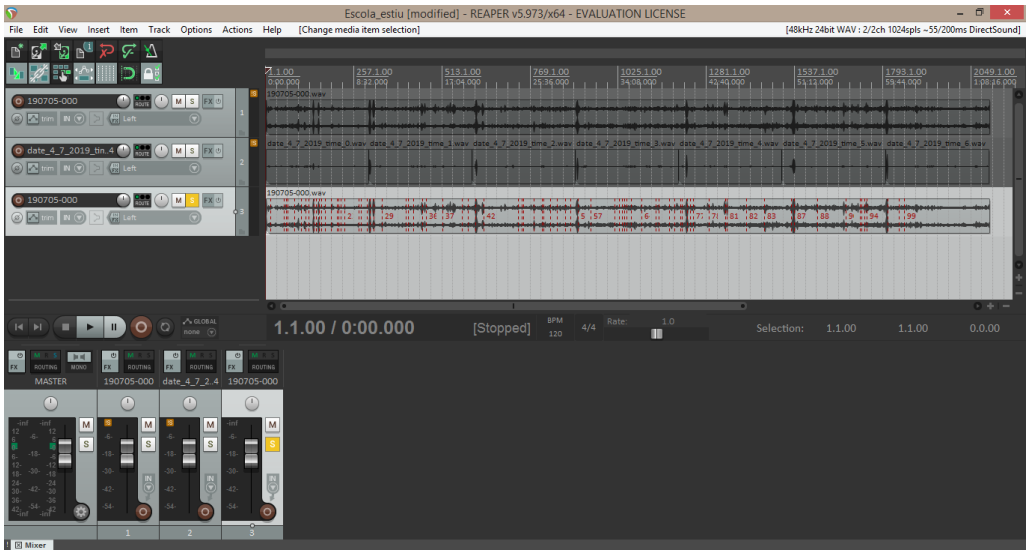


Figura 2.9: Interfaz de usuario de *Reaper*. Fuente: Elaboración propia.

2.4.2. Metodología

Se considera importante hablar de la metodología que se debería de seguir para realizar esta labor, ya que, como bien es sabido, el trabajo de obtención de datos para entrenar un clasificador es bastante tedioso [1]. Además de captar dichos datos, es necesario verificar que sean válidos y que no sean erróneos y/o ambiguos. Es necesario antes de clasificar realizar una escucha de las grabaciones, esta labor es útil para recordar los eventos sucedidos en las ubicaciones y considerar si es necesario añadir alguna clase de ruido a las establecidas *a priori*. Ser más concreto con la taxonomía y añadir nuevas clases lleva más tiempo pero consigue que se puedan hacer más distinciones y obtener una base de datos de fuentes de ruido más diversificada.

En primer lugar se explicará la metodología que se ha seguido en un principio en la tarea de segmentación de tomas y en segundo lugar se explicará otra que se ha realizado al final siendo más óptima que la anterior.

En la tarea de segmentado, en un principio se había optado por escuchar toda la grabación e ir segmentando según se encuentren distintos eventos o según se encuentre que se escuche una distinta fuente de ruido predominante en la grabación. Una vez se han segmentado los archivos de audio, se centra atención en una clase o tipo de fuente de ruido y se mueven los segmentos que pertenecen a dicha clase a las pistas 4 y 5. Una vez se considera que hay una cantidad significativa de segmentos, se renderizan los fragmentos a la carpeta de la clase a la que pertenecen. Se prosigue hasta terminar con todos los fragmentos del proyecto. Como se observa es una metodología bastante tediosa dado a que obliga a escuchar todos los fragmentos varias veces.

La alternativa a la que se ha llegado es utilizar otra metodología más eficiente. En este caso lo que se hizo es con la grabación segmentada (se mantiene la tarea de segmentar la grabación antes de clasificar), generar tantas pistas como clases de ruido que se tengan en cada ubicación, generando paralelamente las pistas necesarias. De esta manera, se reduce considerablemente el número veces que es necesario escuchar un segmento, se ubican los pares de segmentos en la pista que corresponde a su clase y agiliza la tarea de redenderizar luego. En la Figura 2.10 se muestra una captura de cómo se han organizado las pistas de *Reaper* para aplicar esta metodología. Este método resulta más cómodo y ordenado que el inicial para renderizar las tomas.

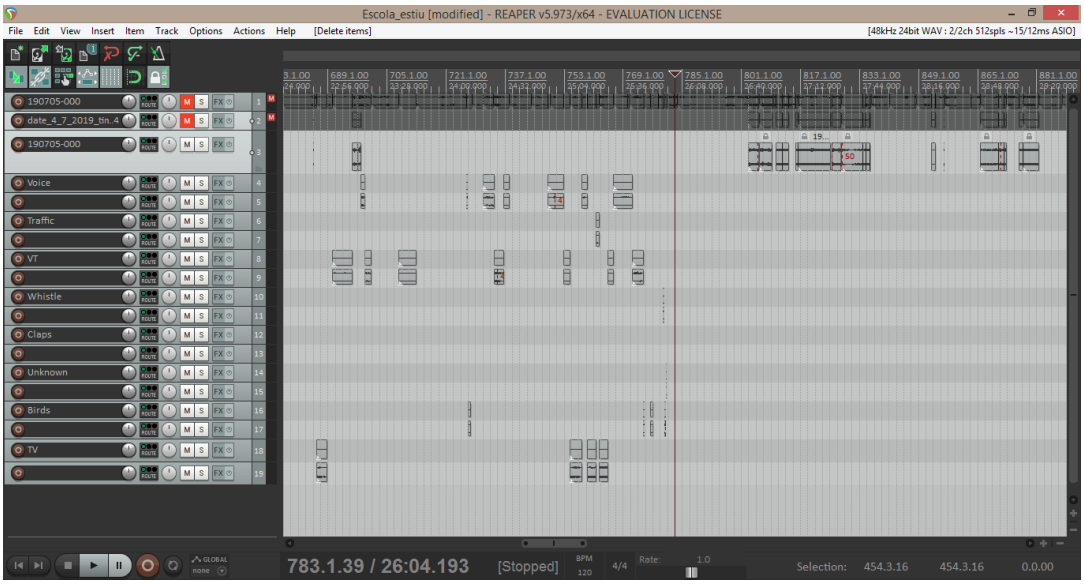


Figura 2.10: Nueva metodología de clasificación. Fuente: Elaboración propia.

2.5 Organización de la base de datos

El etiquetado de los fragmentos de audio se ha basado en crear un directorio que contendrá todos los fragmentos de grabaciones renderizados que pertenecen a la misma clase. Se hace en paralelo con los archivos de la grabadora y con los de la Raspberry.

Tras haber realizado la segmentación se ha determinado que finalmente se clasificarían los segmentos en 10 clases:

1. ***Birds***. Pájaros.
2. ***Claps***. Palmadas.
3. ***Horn***. Claxon.

Tabla 2.3: Información sobre los fragmentos etiquetados de la base de datos.

Clase	Número de archivos	Duración
Birds	88	1 minuto y 58 segundos
Claps	58	31 segundos
Horn	79	1 minuto y 13 segundos
Siren	5	52 segundos
Traffic	541	1 hora, 11 minutos y 45 segundos
Traffic+Voices	116	8 minutos y 8 segundos
Unknown	163	5 minutos
Voices	392	59 minutos y 23 segundos
Voices+Traffic	147	10 minutos y 25 segundos
Whistle	40	30 segundos

4. ***Siren***. Sirena de ambulancia.
5. ***Traffic***. Tráfico de vehículos motorizados.
6. ***Traffic+Voices***. Tráfico con voz de fondo.
7. ***Unknown***. Fuente desconocida.
8. ***Voices***. Voz.
9. ***Voices+Traffic***. Voz con ruido de tráfico de fondo.
10. ***Whistle***. Silbato.

Se crean las clases ***Voices+Traffic*** y ***Traffic+Voices*** para denotar voz con ruido de tráfico de fondo y tráfico con ruido de voz de fondo respectivamente. Esto se ha realizado porque se han encontrado tanto muestras de tráfico como de voz que estaban contaminadas con estos ruidos de fondo.
La clase *Unknown* se utiliza para encasillar las muestras que no se pueden catalogar en la taxonomía planteada porque o bien contienen ruido procedente de una fuente que no se tiene en cuenta en la creación de la base de datos o porque plantea una ambigüedad elevada entre clases de la taxonomía. También se tiene pensado utilizar esta clase para encasillar todas las muestras de alguna clase que genere problemas con el modelo (que la probabilidad de detectar dicha clase sea baja).

En la Figura 2.11 se muestra cómo se han organizado los directorios en la base de datos, el directorio *1* corresponde a la base de datos de la grabadora y el directorio *2* corresponde con la base de datos creada con las grabaciones de la Raspberry. En la Tabla 2.3 se muestran los archivos de audio que hay para cada clase y la duración total del conjunto de estos datos, son los mismos archivos y duraciones tanto para la base de datos de las grabaciones de la grabadora como de las grabaciones realizadas con la Raspberry.

En esta base de datos, el directorio raíz corresponde a la carpeta *Classification*, tal y como se observa en la Figura 2.11, después de esta carpeta se encuentran las carpetas *Database* y *Features*. En la carpeta *Database* se guardan todos los fragmentos de audio clasificados y en la carpeta *Features* se guardarán las *features* que se generen de cada archivo de audio que se encuentre en *Database*.

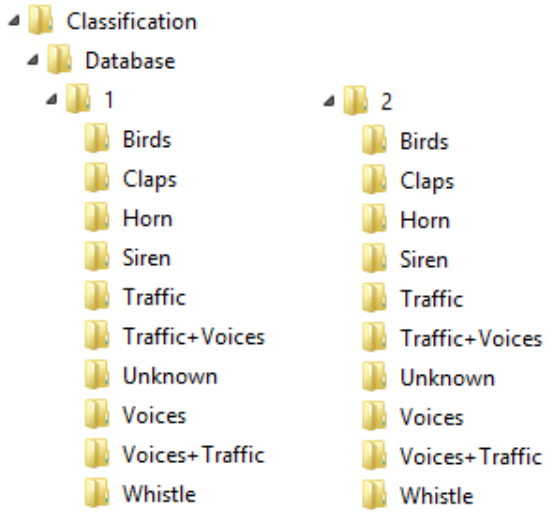


Figura 2.11: Organización de los directorios de la base de datos de ruidos. Fuente: Elaboración propia.

CAPÍTULO 3
Elección de *features*

Cuando ya queda lista la base de datos, se procede a extraer las *features* de los segmentos de audio. Para ello se implementan un conjunto de funciones de MATLAB que se encargan de: Leer los segmentos de una clase, extraer las *features* de cada segmento y finalmente, guardar los vectores que genere cada fragmento de audio en un archivo de extensión .mat. Como ya se mencionado en la Sección 2.5, las *features* se guardan en un conjunto de directorios similar al que aparece en la Figura 2.11, lo único que cambia es que en lugar de la carpeta *Database*, se trabaja sobre la carpeta *Features*, siendo los subdiretorios de esta similares al árbol de directorios al que se ha hecho referencia.

Siguiendo la jerarquía planteada en la Subsección 1.3.1, las tomas con una duración superior a 4 segundos habrá que dividirlas en segmentos de 4 segundos, siendo el último de una duración igual o inferior. Así pues, de cada segmento se extraerá un vector de *features*, pudiendo cada toma generar más de un vector. La función que se ha implementado en MATLAB ya se encarga de hacer este troceado antes de extraer el vector de *features*.
Dado a que las señales de audio no suelen ser estacionarias y esto lleva a que la señal tenga tendencia a variar sobre el tiempo, se utiliza una técnica de procesado llamada *short-term*. En esta técnica, las muestras se la señal de audio que conforma el segmento se descomponen en un número de tramas y se realiza el análisis a nivel de trama [2, 15]. En el tiempo de duración de trama se podría considerar la señal de audio como estacionaria en el tiempo pero esta duración debe de ser la suficiente como para determinar correctamente los parámetros de interés.

De todas las tramas que conforman un segmento se obtiene un vector de *features*. Sobre el vector obtenido, se calculan estadísticas (media, varianza, máximo, mínimo, etc), dependiendo del caso, que llevarán a obtener una *feature* estadística para cada segmento. Esta técnica se conoce como *mid-term processing* [2]. En la Figura 3.1 se muestra de una manera gráfica como se realiza este procesado sobre los segmentos.

Antes de proceder con la extracción de las *features* para formar un *dataset* propio, hay que hacer un pre-procesado de los segmentos. Para empezar, convierte los segmentos que pudiesen estar en estéreo a mono calculando el promedio de ambos canales de audio (en este caso no es necesario porque ya se han guardado las tomas en mono), se elimina la componente en contínua de la señal tal y como se hace en [16]. Después, se normaliza la señal en potencia y se fijan longitudes de las tramas en un valor fijo con un solape (*overlap*) del 50 %. La extracción de características se realizará para tamaños de trama de 1024, 2048 y 4096 muestras, teniendo así 3 directorios que contendrán las *features* correspondientes a cada longitud de trama. Posteriormente se elegirá qué longitud de trama será la más adecuada para el clasificador. En la Tabla 3.1 se observa la relación entre el número de muestras y la duración de una trama para una frecuencia de muestreo de 48 kHz. Una vez se ha fijado esto, se procede a extraer las *features*.

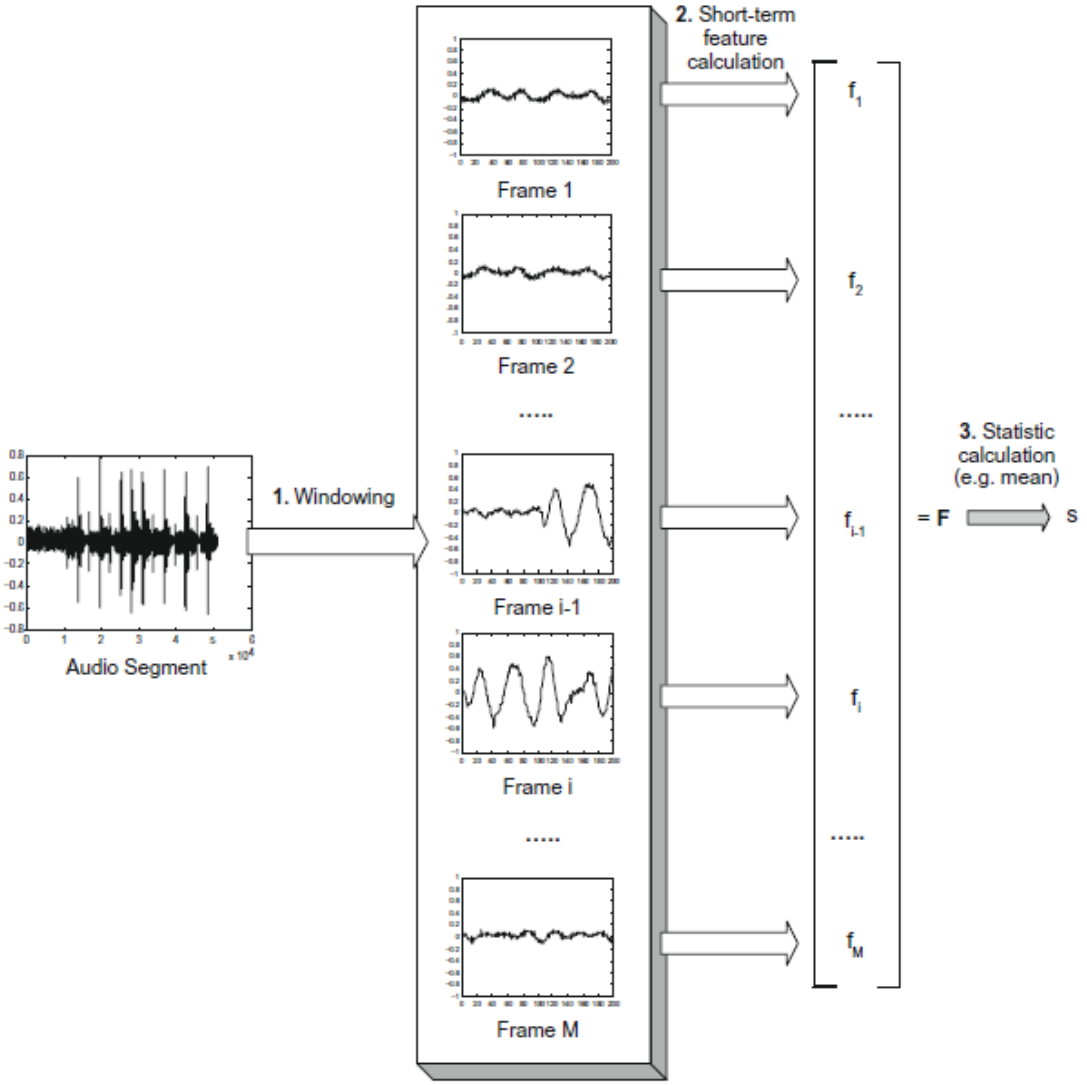


Figura 3.1: *Mid-term feature extraction*: cada segmento se divide en tramas para hacer el proceso sobre éstas y se realiza una estadística sobre la secuencia de *features* que han generado las tramas del segmento. Fuente: *Introduction to Audio Analysis: A MATLAB Approach* [2].

Para el proceso de extracción de las *features* espectrales y de los MFCC (*Mel Frequency Cepstral Coefficients*) se han utilizado las funciones del *Audio Toolbox* de MATLAB [17], en cambio, para extraer las *features* temporales se han utilizado las funciones que aparecen en [16]. En la Figura 3.2 se muestra cómo las funciones del *Audio Toolbox* de MATLAB extraen las características espectrales y los MFCC de los segmentos de audio.

3.1 *Features* temporales

Normalmente, las *features* temporales se obtienen utilizando las muestras de la señal de audio [2], la señal se procesa en el tiempo. Igual que como se ha realizado en [16], para generar las tramas se utiliza una ventana rectangular de 1024 muestras (21.333 ms) con un 0 % de solapamiento.

Para la obtención de las características temporales de los segmentos se han utilizado las funciones que venían implementadas en [16].

Tabla 3.1: Duraciones de las tramas en muestras y milisegundos para una frecuencia de muestreo de 48 kHz.

Tamaño en muestras	Duración
1024	21.33 ms
2048	42.67 ms
4096	85.33 ms

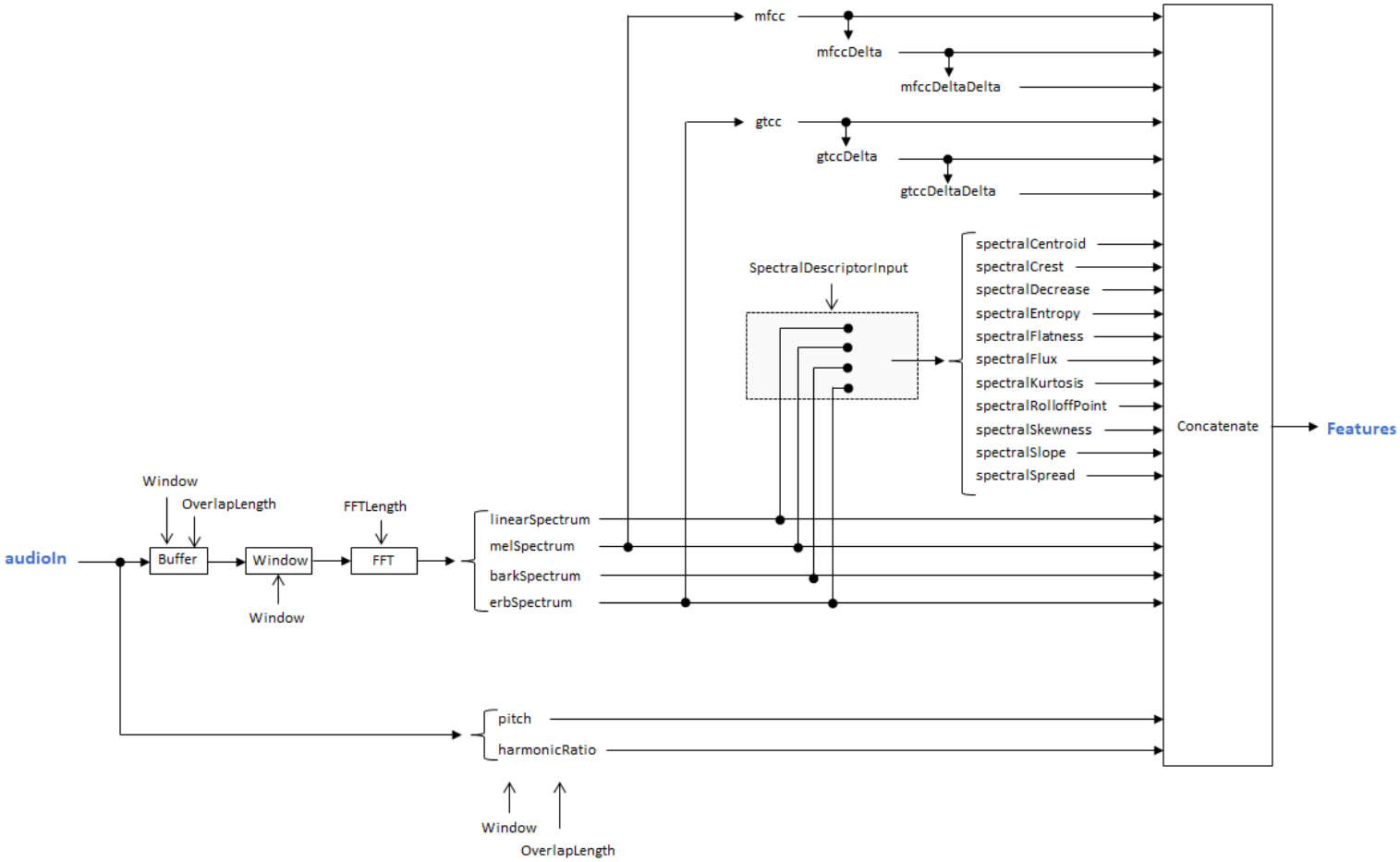


Figura 3.2: Proceso que sigue un segmento de audio para la extracción de las *features* espectrales y cepstrales. Fuente: [18]

3.1.1. *Zero Crossing Rate*

El *Zero Crossing rate* (ZCR) de una trama es la cantidad de veces que la señal cambia de signo durante la trama. Dicho de otra manera, las veces que la señal pasa por cero [2, 3]. El ZCR puede dar una idea de lo ruidosa que es una señal, ya que, una trama que contenga una presencia mayor de ruido, tendrá un valor más elevado en esta *feature* [2]. En (3.1) se muestra la expresión para obtener esta feature en una trama, siendo ZCR el valor a obtener, W_L el tamaño de la ventana que se ha utilizado para obtener la trama, $x(n)$ la muestra n-ésima de la trama y $sgn(.)$ la función signo de un número [2].

$$ZCR = \frac{1}{2W_L} \sum_{n=1}^{W_L} |sgn[x(n)] - sgn[x(n-1)]| \tag{3.1}$$

3.1.2. Índice de modulación AM

Para el cálculo de esta característica del audio se divide la señal en tramas, en cada trama se obtiene la envolvente de esta señal calculando el valor máximo y promediando todos los valores dentro de la trama, tal y como se muestra en (3.2). Una vez se ha aplicado este procedimiento en todas las tramas, se calcula la media de los índices de modulación que se obtienen. Las señales de voz dan como resultado un índice de modulación alto porque su envolvente contiene cambios abruptos, en cambio, señales como las de ruido de tráfico, donde la envolvente es más estable, da como resultados índices de modulación más pequeños [16].

$$m(\,\%) = 100\frac{I_{max} - I_0}{I_{max}} \tag{3.2}$$

3.1.3. *Log-Attack Time*

Esta *feature* corresponde al tiempo de ataque de una señal después de que se le aplique un logaritmo en base 10. Se establecen los puntos en los que empieza y termina el ataque de una señal acústica, estableciendo como inicio el 25 % del valor RMS de la señal [16]. En (3.3) se muestra cómo se calcula esta característica. t_{stop} es el instante en el cual termina el ataque de la señal y t_{inicio} es el instante en el que empieza el ataque.

$$LogAttack = log_{10}(t_{stop} - t_{inicio}) \tag{3.3}$$

3.1.4. *Pitch*

Las señales pueden ser clasificadas como cuasiperiódicas o aperiódicas. Las señales cuasiperiódicas muestran tener un comportamiento periódico, serían señales acústicas procedentes de música, de una sirena o un cláxon. Por otra parte, las señales aperiódicas son las que tienen un comportamiento similar al fuido, estas señales en general procederían del tráfico, aplauso o ruido [2].

El *pitch* representa la frecuencia percibida de un determinado sonido. En la literatura, este término se considera equivalente al de frecuencia fundamental, aunque no sea exactamente lo mismo, en muchos casos la frecuencia fundamental y el *pitch* coinciden, de ahí esta simplificación [2]. Para obtener la frecuencia fundamental existen varias técnicas, en este trabajo, se obtendrá el pitch mediante la función de autocorrelación [16]. Con este método, se calcula la correlación que tiene una señal con respecto a la misma señal desplazada un número determinado de muestras, si la señal es cuasiperiódica, esta correlación tomará un valor alto para un determinado desplazamiento (periodo). En cambio, si la señal es aperiódica, la correlación tomará un valor alto cuando el desplazamiento de la señal sea nulo [2, 16].

Para la extracción del pitch se utilizará la función de autocorrelación normalizada (3.1), la cual se pasará a llamar $\Gamma(m)$. Luego, se obtiene el máximo de $\Gamma(m)$, también conocido como rango armónico. Finalmente, el periodo fundamental se selecciona como la posición donde se encuentra el máximo de $\Gamma(m)$ [2].

$$\Gamma(m) = \frac{\sum_{n=1}^{W_L} x(n)x(n-m)}{\sqrt{\sum_{n=1}^{W_L} x(n)^2 \sum_{n=1}^{W_L} x(n-m)^2}} \tag{3.4}$$

3.2 *Features* espectrales

Estas *features* se basan en la Transformada Discreta de Fourier (DFT en inglés) de las tramas. En esta trabajo, se calcula la DFT a través de la Transformada Rápida de Fourier (FFT en inglés), la cual aprovecha la redundancia computacional en las expresiones que definen la DFT. Esta transformación da como resultado la representación del contenido en frecuencia de un sonido [2].

Como ya se ha detallado al principio del capítulo, las funciones del *Audio Toolbox* de MATLAB que extraen las *features* espectrales enventanando las tramas que conforman el segmento con una ventana *Hamming* aplicando un solape del 50 % y se realiza la FFT. Con el espectro que se obtiene tras realizar la FFT se obtienen las características espectrales.

3.2.1. *Slope*

La pendiente espectral o el *Slope* describe cómo decae la amplitud espectral de la trama. Es resultado de calcular la regresión lineal sobre el espectro obtenido de la trama. Se obtiene como resultado la pendiente de una recta que describe cómo decae el espectro de una señal a través de los distintos valores de frecuencia [16].

Para obtener esta *feature* se ha utilizado la función `spectralSlope`. La expresión que se utiliza en esta función viene dada por (3.5), donde f_k es la frecuencia en hercios del *bin* k, s_k es el valor espectral de la señal en el bin k, μ_f es la frecuencia media, μ_s el valor espectral medio, y b_1 y b_2 son las bandas en *bins* sobre las que se calculará el *slope* [19].

Un *bin* es un segmento del eje de frecuencias que contiene la amplitud de un rango pequeño de frecuencias obtenidas como resultado de la DFT. Se puede interpretar como una muestra tomada del dominio de la frecuencia que dependerá de la resolución de la DFT.

$$slope = \frac{\sum_{k=b_1}^{b_2}(f_k - \mu_f)(s_k - \mu_s)}{\sum_{k=b_1}^{b_2}(f_k - \mu_f)^2} \tag{3.5}$$

3.2.2. *Decrease*

De una manera similar a *Slope*, el decaimiento espectral o el *Decrease* representa la cantidad que decae el espectro, con la diferencia de que se hace un mayor enfásis en las bajas frecuencias, pareciéndose más a la percepción humana. Se suele utilizar junto al *Slope* [19, 16].

Para obtener esta *feature* se ha utilizado la función `spectralDecrease`. La expresión que se utiliza en esta función viene dada por (3.6), donde s_k es el valor espectral de la señal en el bin k, y b_1 y b_2 son las bandas en *bins* sobre las que se calculará el *decrease* [19].

$$decrease = \frac{\sum_{k=b_1+1}^{b_2} \frac{s_k - s_{b_1}}{k-1}}{\sum_{k=b_1+1}^{b_2} s_k} \tag{3.6}$$

3.2.3. *Flux*

La variación espectral, o *flux*, mide los cambios espectrales que se producen entre tramas consecutivas calculando la diferencia cuadrática entre las magnitudes normalizadas del espectro, es una medida cómo varía el espectro con respecto al tiempo. Un valor alto

en la variación espectral indica que hay más cambios espectrales a nivel local. Por ejemplo, una señal de voz tiene un valor de *flux* superior al de una señal de música, esto se debe a los cambios de fonemas [2] [19].

Para obtener la variación espectral se ha utilizado la función `spectralFlux`. La expresión que se utiliza en esta función viene dada por (3.7), donde s_k es el valor espectral de la señal en el bin k, p el tipo de normalización, y b_1 y b_2 son las bandas en *bins* sobre las que se calculará el *flux* [19].

$$flux(t) = (\sum_{k=b_1}^{b_2} |s_k(t) - s_k(t-1)|^p)^{\frac{1}{p}} \tag{3.7}$$

3.2.4. *Roll-Off*

Es la frecuencia en la cual un porcentaje de la energía del espectro está por debajo de la energía total, teninedo el espectro como una distribución de probabilidad [16]. Esta característica se puede utilizar para distinguir entre fonemas sonoros y sordos o entre distintos tipos de música [2]. En este caso, se ha ajustado el porcentaje al 95 %.

En el cálculo del *Roll-Off* se ha utilizado la función `spectralRolloffPoint`. La expresión utilizada en esta función es la que figura en (3.8), donde s_k es el valor espectral de la señal en el bin k, u es el umbral de energía que se especifica (normalmente 85 % o 95 %), y b_1 y b_2 son las bandas en *bins* sobre las que se calculará esta *feature* [19].

$$RollOff = i, talque \sum_{k=b_1}^{i} |s_k| = k \sum_{k=b_1}^{b_2} s_k \tag{3.8}$$

3.2.5. *Centroid*

El *centroid*, o centroide espectral, aporta información acerca de la forma del espectro. El centroide espectral es el centro de gravedad del espectro, o lo que es lo mismo, la frecuencia media del espectro ponderando en amplitud [2, 16]. Un valor alto del centroide está relacionado con sonidos más brillantes [2, 19].

Para el cálculo del centroide espectral se ha utilizado la función `spectralCentroid`. Esta función utiliza la expresión (3.9), donde f_k es la frecuencia en hercios del *bin* k, s_k es el valor espectral de la señal en el bin k, y b_1 y b_2 son las bandas en *bins* sobre las que se calculará el centroide espectral [19].

$$centroide = \frac{\sum_{k=b_1}^{b_2} f_k s_k}{\sum_{k=b_1}^{b_2} s_k} \tag{3.9}$$

3.2.6. *Spread*

La extensión espectral, o *spread*, mide la distribución del espectro alrededor del centroide. Valores bajos en el *spread* indican que el espectro está más concentrado alrededor del centroide espectral. Se toma como la desviación estándar del espectro del centroide [2].

Para calcular la extensión espectral se utiliza la función `spectralSpread`. La expresión que utiliza esta función viene dada por (3.10), donde f_k es la frecuencia en hercios del *bin*

k, s_k es el valor espectral de la señal en el bin k, μ_1 es el centroide espectral, y b_1 y b_2 son las bandas en *bins* sobre las que se calculará el *spread* [19].

$$spread = \sqrt{\frac{\sum_{k=b_1}^{b_2}(f_k - \mu_1)^2 s_k}{\sum_{k=b_1}^{b_2} s_k}} \tag{3.10}$$

3.2.7. *Skewness*

La asimetría espectral o *skewness* es una medida de la simetría de la energía espectral alrededor del centroide, indicando también, en caso de no haber simetría, si hay más energía en frecuencias superiores o inferiores al centroide [16] [19].

La asimetría espectral se calcula en la función `spectralSkewness` utilizando la expresión (3.11), donde f_k es la frecuencia en hercios del *bin* k, s_k es el valor espectral de la señal en el bin k, μ_1 es el centroide espectral, μ_2 es la extensión espectral, y b_1 y b_2 son las bandas en *bins* sobre las que se calculará esta característica [19].

$$curtosis = \frac{\sum_{k=b_1}^{b_2}(f_k - \mu_1)^3 s_k}{\mu_2^3 \sum_{k=b_1}^{b_2} s_k} \tag{3.11}$$

3.2.8. *Kurtosis*

Si se toma el espectro de una señal como una variable aleatoria, un valor alto de curtosis indica que el espectro presenta más picos alrededor del centroide, en cambio, un valor bajo indica que el espectro es más plano alrededor del centroide [16]. Indica cómo de plano es el espectro alrededor del centroide de este. Por ejemplo, si en una toma en la que hay voz se aumenta el sonido blanco, la curtosis decrece [19].

La curtosis espectral se obtiene a partir de la función `spectralKurtosis`. Esta función utiliza la expresión (3.12), donde f_k es la frecuencia en hercios del *bin* k, s_k es el valor espectral de la señal en el bin k, μ_1 es el centroide espectral, μ_2 es la extensión espectral, y b_1 y b_2 son las bandas en *bins* sobre las que se calculará la curtosis espectral [19].

$$curtosis = \frac{\sum_{k=b_1}^{b_2}(f_k - \mu_1)^4 s_k}{\mu_2^4 \sum_{k=b_1}^{b_2} s_k} \tag{3.12}$$

3.3 MFCC

Los *Mel Frequency Cepstrum Coefficients* (MFCC) permiten representar la señal en el dominio cepstral habiendo distribuido sus bandas de frecuencias siguiendo la escala Mel en lugar del espaciado lineal. El dominio cepstral se obtiene realizando la transformada inversa de Fourier (IFT) del logaritmo del espectro de la señal [16].

Primero se cálcula la Transformada Discreta de Fourier (DFT) de la señal de entrada. El espectro obtenido de la señal después de aplicar la DFT servirá como entrada a un banco de L filtros que siguen la escala Mel. Desde el punto de visto de la psicoacústica, el oído humano tiene la capacidad de distinguir frecuencias cercanas más fácilmente en las bajas frecuencias que en las altas, por lo tanto, la escala Mel introduce un efecto que deforma la frecuencia para imitar el comportamiento del oído humano. La expresión de esta deformación en frecuencia viene dada por (3.13), siendo f_ω el valor de la frecuencia

f en esta escala. Simplificando, la escala Mel utiliza intervalos de frecuencia los cuales el oído humano percibiría como equiespaciados [2].

$$f_\omega = 1127{,}01048 \cdot log(\frac{f}{700} + 1) \tag{3.13}$$

Entonces, definiendo $\widetilde{O}_k$ como la potencia de salida del filtro k-ésimo del banco de filtros, entonces los coeficientes resultantes y correspondientes a la MFCC vedrían dados por la expresión (3.14). De acuerdo a esta expresión, los MFCCs son coeficientes de la transformada discreta del coseno del espectro en escala Mel en términos de potencia logarítmica [2].

$$c_m = \sum_{k=1}^{L} (log\widetilde{O}_k) cos[m(k - \frac{1}{2})\frac{\pi}{L}], m = 1, ..., L. \tag{3.14}$$

Se ha demostrado que los MFCCs son muy buenas *features* en varias aplicaciones de análisis de audio, entre ellas, para distinguir entre música y el habla.

Los MFCCs se han utilizado para aplicaciones de reconocimiento del habla, para separar a las diferentes personas que hablan en una misma conversación y muchas otras aplicaciones más. En el caso de la tarea de distinguir entre habla y música, la capacidad discriminativa de estas features es bastante alta, se estima un error Bayesiano equivalente a un 11.8 % utilizando sólamente los MFCC como *features*.

Depende de la tarea que se quiera realizar se utilizan diferentes subconjuntos de los MFCCs. Por ejemplo, en aplicaciones en las que se procesa música se suelen seleccionar los primeros 13 MFCCs, ya que, llevan información suficiente en el contexto de varias tareas de clasificación [2].

También se utilizan la primera y segunda derivada de los coeficientes MFCC obtenidos de cada trama (Δ-MFCC y ΔΔ-MFCC respectivamente) las cuales aportan información de la variación de estos coeficientes con respecto al tiempo [16].

Al obtener todos los coeficientes MFCC de las tramas que conforman un segmento, se calculan una serie de estadísticas por cada conjunto de coeficientes i-ésimos. De esta manera, al calcular una estadística de los MFCC, se calcula la estadística para cada conjunto de coeficientes i-ésimos, obteniendo un vector de esta estadística de igual longitud al vector de coeficientes MFCC. Las estadísticas que se aplican sobre los MFCC son: el mínimo, el máximo, la media, la mediana, la curtosis y la asimetría estadística (*skewness*). [4].

Los coeficientes MFCC se han obtenido como resultado de utilizar la función función `mfcc` introduciendo el segmento como entrada, la longitud de la trama, el solape que se va a utilizar y el número de coeficientes que se quiere obtener [20].

3.4 Extracción de *features*

Después de que se hayan extraído las *features* se obtienen, tal y como se observa en la Tabla 3.2, los vectores de *features* por segmento para cada clase. Como hay tomas que tienen una duración superior a 4 segundos, de ellas se obtiene más de un vector de *features*, por eso, se observa que el número de features que se obtiene es superior al número de fragmentos de audio.

Para extaer las características de los archivos se han utilizado *scripts* que se han implementado en MATLAB y las funciones encargadas de extraer las diferentes features

Tabla 3.2: Features extraídas para cada clase.

Clase	Vectores de *features*
Birds	91
Claps	58
Horn	79
Siren	15
Traffic	1373
Traffic+Voices	187
Unknown	191
Voices	1103
Voices+Traffic	234
Whistle	40

temporales, espectrales y cepstrales. En las figuras 3.3 y 3.4 se describe el funcionamiento de las funciones implementadas para esta tarea.
Este conjunto de *scripts* es capaz de extraer todas las *features* de los archivos de un directorio dado y guardarlas en otro directorio que se especifique. Las funciones extraen un total de 301 características de un archivo de audio, siendo las que se han detallado en este capítulo añadiendo también unas *features* espectrales del *Audio Toolbox* de MATLAB [17] que finalmente no se han utilizado en los resultados finales porque no se han encontrado precedentes de que se hayan utilizado dichas características para clasificación de fuentes de ruido urbano, aunque si que se utilizarán en las primeras pruebas sobre el *dataset*. El número de *features* a utilizar en la tarea de clasificación se puede elegir, pudiendo descartar aquellas *features* que no se vayan a utilizar.

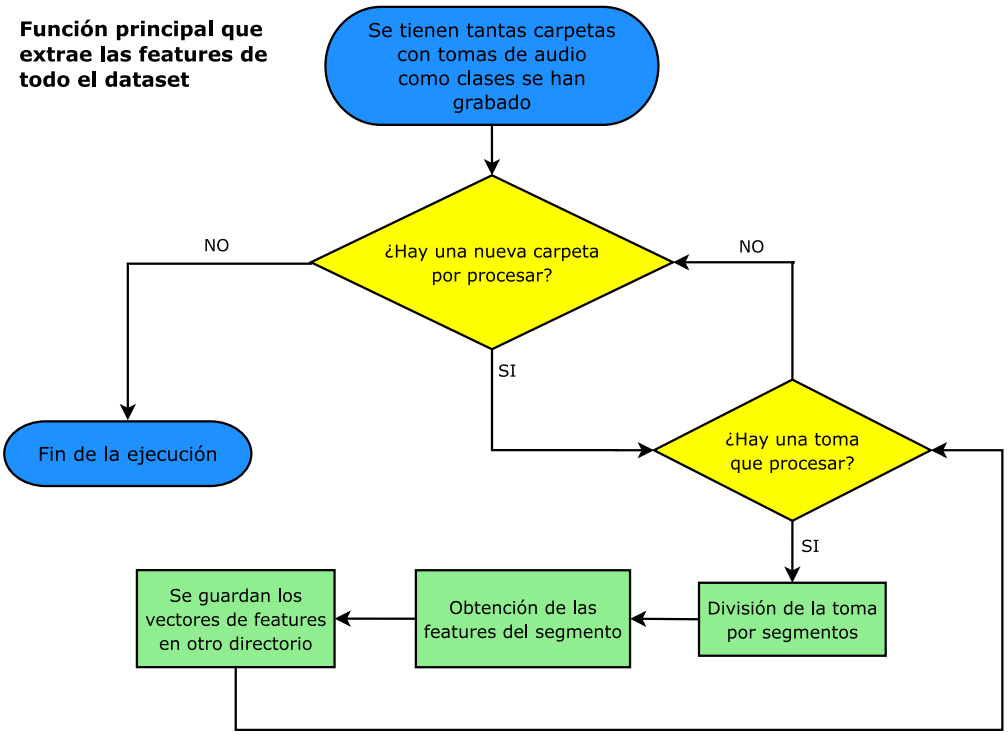


Figura 3.3: Descripción del *script* que guarda los vectores de características. Fuente: Elaboración propia.

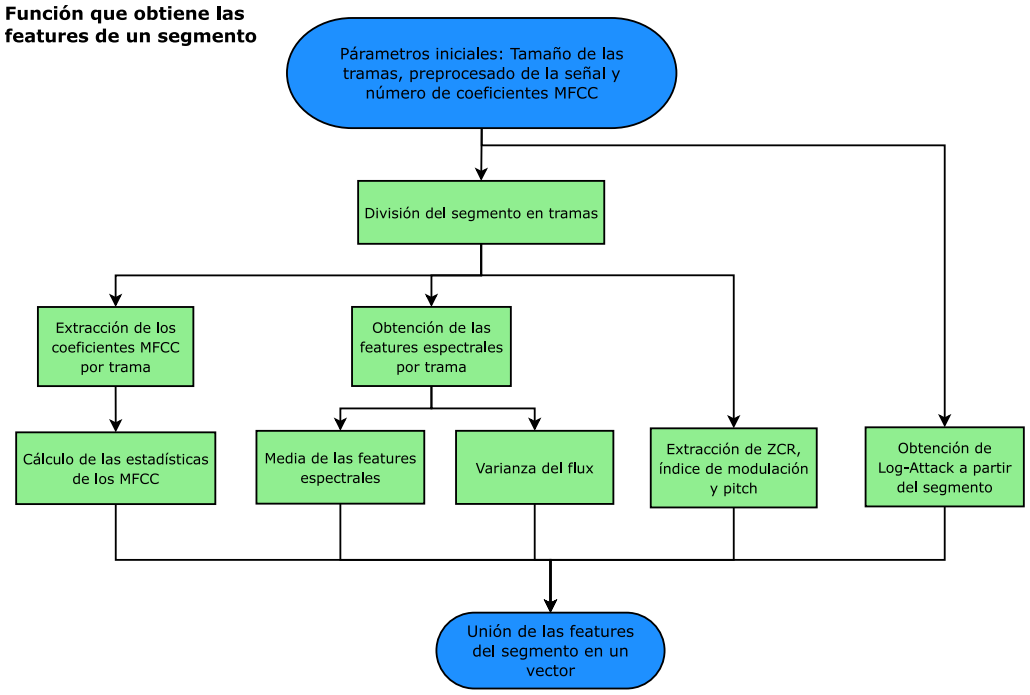


Figura 3.4: Descripción de la función que extrae las *features*. Fuente: Elaboración propia.

Por último, cabe añadir que el proceso de extracción de *features* ha sido el que más coste computacional ha tenido, o dicho de otra manera, el que más ha tardado en realizarse, ya que, se ejecuta sobre toda la base de datos.

CAPÍTULO 4
Clasificadores

En el Capítulo 3 se ha mostrado el proceso que se ha llevado a cabo para extraer las *features* de los segmentos de audio que conforman la base de datos. Una vez ya se tienen estas características del audio, un clasificador podría ser capaz de distinguir entre una clase de sonido u otra observando los valores que toman estas *features*.

4.1 Tipos de clasificadores

Para este trabajo, basándose en [16] y [4], se han elegido los modelos de aprendizaje supervisado *Random Forest* y *Support Vector Machine*, ya que, son los que han demostrado obtener los mejores resultados en las tareas de clasificación de fuentes de sonido.

4.1.1. *Random Forest*

El algoritmo *Random Forest* o también llamado Bosques Aleatorios es un algoritmo que se basa en utilizar múltiples árboles de decisión para realizar una tarea de clasificación[16].

Dentro de los árboles de decisión se encuentran dos categorías: Los árboles de regresión, cuya salida es un número real, y los árboles de clasificación, cuya salida es la etiqueta que se da a una cierta clase, siendo éstos los que se utilizarán en este trabajo. El funcionamiento de un árbol de decisión se basa en realizar decisiones secuenciales sobre características individuales. El árbol está compuesto por una serie de nodos, cada nodo del que se compone el árbol representa una pregunta con respuesta positiva o negativa. Esta pregunta se basa en si el valor de una *feature* en concreto es superior a un valor determinado. De cada nodo salen dos ramas que representan la respuesta positiva o negativa a esta pregunta. Estas ramas llevan a otro nodo del cual desembocan otras ramas. Finalmente, las últimas ramas llevan a las hojas, siendo cada hoja la asignación a una clase determinada. En la salida de cada hoja se obtiene como resultado un histograma que representa la probabilidad que tiene la muestra de pertenecer a cada clase, siendo la más problable la clase que se asigne en caso de utilizar un árbol de decisión [2, 16].

Entonces, una muestra de la cual se quiere predecir a qué clase pertence, entra por el nodo principal del árbol (o nodo raíz), se mueve a través de los nodos secundarios y las ramas hasta llegar a una hoja. Al llegar al final del árbol, el árbol predice a qué clase pertenece esta muestra. Se debe tener en cuenta que en un árbol pueden haber características de las muestras que no participen en las decisiones que se tomen sobre estas. En la fase de entrenamiento se determina el umbral de cada nodo que mejor separa entre dos clases diferentes [2, 16].

Una vez se entiende en qué consiste un árbol de decisión, el algoritmo *Random Forest* es una manera de ensamblar un número determinado de árboles de decisión entrenados. Cada árbol se entrena con subconjunto de características que son asignadas de manera aleatoria. La predicción de una muestra de *test* será el correspondiente de realizar un promedio de los histogramas que se obtienen de las hojas de todos los árboles que forman el bosque aleatorio. El resultado de la predicción será la clase con mayor probabilidad. Se utilizan 500 árboles de decisión para implementar el *Random Forest* [16].

4.1.2. *Support Vector Machine*

Si se tienen las muestras pertenecientes a 2 clases diferentes situadas en un espacio de n dimensiones (cada dimensión es una *feature*), el *Support Vector Machine* o también llamado por sus siglas SVM, buscará un hiperplano de decisión que separe ambas clases y que maximice el margen de las muestras con respecto a éste, dando como resultado el menor error de clasificación en las muestras que se utilizan para entrenamiento. En un lado del hiperplano se obtendrá un valor mayor que 1 (pertenece a una clase) o menor que 1 (no pertenece a esa clase) [16] [2].

Cuando la curva que debería de separar a ambas clases no es lineal, las clases no quedan del todo separadas o se desea distinguir entre más de 2 clases, se utilizan funciones no lineales para establecer el margen de decisión entre varias clases, estas funciones se llaman *kernels*. Los *kernels* son funciones que proyectan los valores de las *features* de cada muestra en un nuevo espacio. Esta proyección permite que para el clasificador sea más sencillo encontrar un hiperplano que sea capaz de separar entre las muestras de distintas clases [1].
Como se ha hecho en [16] se ha utilizado un *kernel* polinómico de tercer orden. $G(x_j, x_k)$ es un elemento de la matriz de *Gram* donde x_k y x_j son vectores que representan los valores de las *features* k y j de cada muestra. La función del *kernel* polinómico (Φ) se aplica sobre x_k y x_j antes de realizar el producto escalar. Todo esto se detalla en la expresión (4.1). El orden de este kernel viene dado en el valor de q que será igual a 3. [21]. La matriz de *Gram* es una matriz nxn donde cada elemento de ésta es el producto escalar de cada columna del *training dataset* después de utilizar la función *kernel*. Con esta matriz, el algoritmo utiliza otro espacio distinto para encontrar el hiperplano que separa las distintas clases [21].

$$G(x_k, x_j) =< \Phi(x_k), \Phi(x_j) >= (x_j' x_k + 1)^q \tag{4.1}$$

4.2 Clasificación multiclase

Cuando el número de clases de salida que se obtiene del clasificador es igual a 2 se afirma que se está realizando una tarea de clasificación binaria. En este tipo de tareas se desea distinguir entre dos clases o si dicha muestra pertenece o no a una clase única. Al haber dos clases, se tiene una salida positiva, la cual se llamará +1, y otra salida negativa, la que se llamará -1. A modo de ejemplo, la implementación de un filtro que distinga entre correo basura y correo convencional, o la implementación de un programa que permita determinar si un tumor es maligno o no a partir de imágenes médicas son problemas de clasificación binaria [1].

Por otro lado, cuando el número de clases que se obtiene a la salida de un clasificador es igual o superior a 3, el clasificador es un clasificador multiclase. En este tipo de problemas se entrenan un conjunto de clasificadores binarios y se combinan sus resultados para

obtener la clase de salida. Esta técnica se conoce como binarización [2]. Entre las técnicas de binarización más conocidas se encuentran *One-vs-One* y *One-vs-All*.

Uno contra uno o *One-vs-One* (OVO), es una técnica en la cual se entrenan $L = k(k-1)/2$ clasificadores binarios, siendo k el número de clases totales que se consideran en el problema de clasificación. Con OVO, se toman las muestras de una clase como salida positiva (+1) y las muestras de otra clase como salidas negativas (-1) de un clasificador binario, las muestras del resto de clases se descartan y no se utilizan en la clasificación (0), este proceso se realiza en todos los clasificadores binarios para todas las combinaciones posibles de pares de clases diferentes [2].
Por otra parte, en la técnica de uno contra todos o *One-vs-All* (OVA), se entrenan $L = k$ clasificadores binarios. Se toman las muestras de una clase como salida positiva (+1) y las muestras del resto de clases como salidas negativas (-1) realizando este proceso para todas las clases. [2]. En la Figura 4.1 se muestra de una forma gráfica cómo trata cada clasificador binario a cada clase en la técnica OVA y en la técnica OVO.

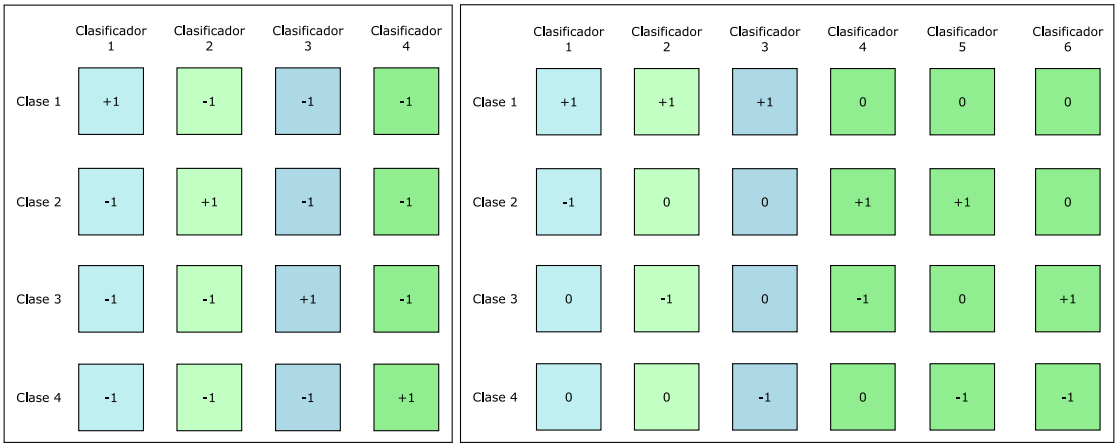


Figura 4.1: Funcionamiento de las técnicas OVA y OVO. A la izquierda se muestra el esquema de funcionamiento de OVA y la derecha el de OVO. Fuente: Elaboración propia

La técnica utilizada para abordar la tarea de clasificación multiclase con clasificadores binarios ha sido *One-vs-One*. Con el algoritmo ya entrenado, se asigna el valor de salida de una muestra a la clase que minimiza la suma de los errores producidos en los L clasificadores binarios, esto se hace en la función `fitcecoc` [21]. Este método se utiliza para un conjunto de clasificadores SVM funcionando cada uno como un clasificador binario, ya que estos clasificadores no disponen de más de dos salidas.

4.3 Validación cruzada

En la tarea de evaluar la precisión de un clasificador, es necesario dividir las muestras que conforman la base de datos en muestras para entrenamiento y muestras de *test*. Se debe tener muestras que no se hayan utilizado en el entrenamiento del clasificador para poder evaluar si éste es capaz de generalizar correctamente cuando se le da una muestra que no ha observado anteriormente. En este trabajo se ha decidido dejar un 80 % de las muestras de cada clase para la fase de entrenamiento del clasificador y un 20 % de las muestras para evaluar la precisión que éste tendrá después del entrenamiento [1].

Para realizar una mejor estimación de la precisión que tendrá el sistema, se realizan un número determinado de pruebas donde en cada prueba las muestras que se seleccionan para el *training* y *test dataset* no van a ser las mismas. De esta manera los resultados de la precisión de cada clasificador serán lo menos dependientes posibles respecto a qué

muestras se elijan en la fase de entrenamiento y de *test* en cada prueba. La precisión del sistema vendrá determinada por la media aritmética de las precisiones que se han obtenido en cada prueba. Las técnicas más conocidas de validación cruzada son: *k-fold cross-validation*, *Leave-one-out cross-validation* y *repeated random sub-sampling validation* [2].

- ***K-fold cross-validation***: Se divide el *dataset* en k subconjuntos diferentes con el mismo tamaño de muestras. El clasificador realiza la fase de entrenamiento y *test* (lo que en este trabajo se ha llamado *prueba*) k veces, utilizando uno de los k subconjuntos para la fase de *test* y el $k-1$ subconjuntos restantes para la fase de entrenamiento. En la Figura 4.2 se muestra el funcionamiento de esta técnica sobre el *dataset*.

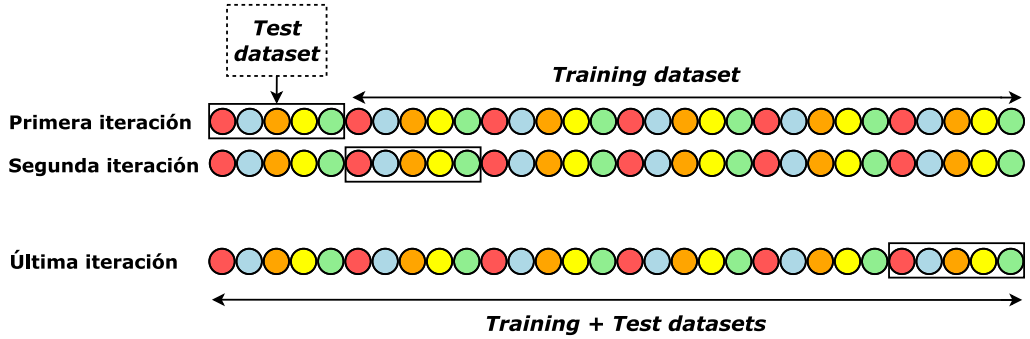


Figura 4.2: Funcionamiento de la técnica *K-fold cross-validation*. Fuente: Elaboración propia.

- ***Leave-one-out cross-validation***: Es un caso del *k-fold* en el cual se hacen tantas particiones como muestras existentes. En cada iteración se utilizan todas las muestras en la fase de entrenamiento excepto una que se utiliza para *test*, repitiendo este proceso k veces. Es el método más costoso computacionalmente. En la Figura 4.3 se muestra el funcionamiento de esta técnica.

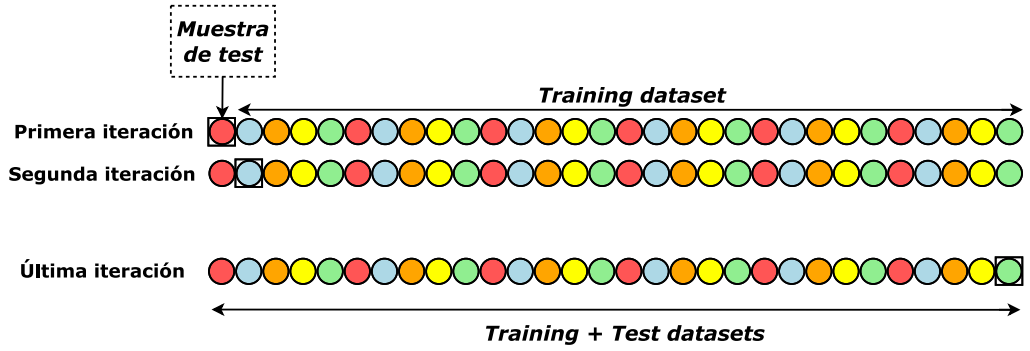


Figura 4.3: Funcionamiento de la técnica *Leave-one-out cross-validation*. Fuente: Elaboración propia.

- ***Repeated random sub-sampling validation***: También se conoce como validación cruzada aleatoria. Para un determinado número de pruebas (o iteraciones), se divide el *dataset* en muestras de entrenamiento y de *test* de manera aleatoria. En este caso, por cada iteración se tomarán, de manera aleatoria, un 80 % de las muestras para la fase de entrenamiento y un 20 % para la fase de *test*. Es posible que en consecuencia al proceso aleatorio algunas muestras no lleguen a estar presentes en el *training* o *test dataset*. En la Figura 4.4 se muestra el funcionamiento de esta técnica.

En este trabajo se ha elegido el método de validación cruzada aleatoria que ha sido el más sencillo de implementar. Esto también ha supuesto otra dificultad dado que se ha

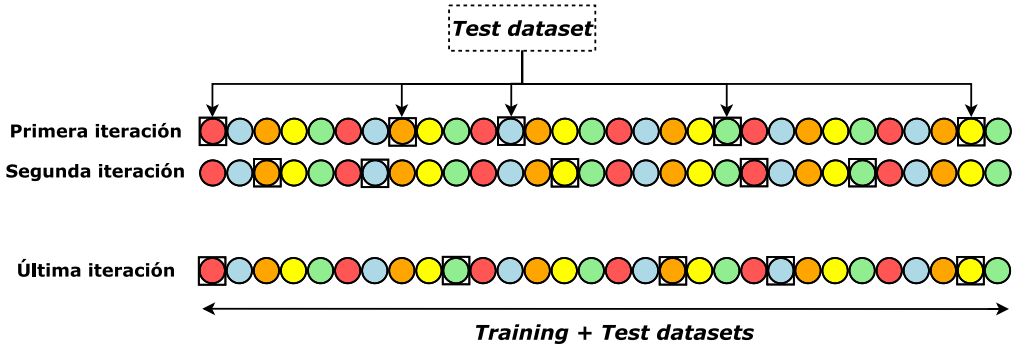


Figura 4.4: Funcionamiento de la Validación cruzada aleatoria. Fuente: Elaboración propia.

trabajado sobre 2 *datasets* de forma paralela. La muestra *i*-ésima del primer *dataset* corresponde a un segmento de audio que se ha grabado con la grabadora y la muestra *i*-ésima del segundo *dataset* corresponde al mismo segmento de audio grabado con la Raspberry. A las muestras que cumplan esta propiedad se llamarán, en este trabajo,primas.
Por lo tanto, se debe de evitar que cualquier muestra de un *dataset* utilizada en la fase de entrenamiento sea prima de cualquier muestra que se utilice en la fase de *test*. Al pertenecer al mismo segmento de audio, podría dar como resultado una precisión mayor de la que realmente tiene el sistema.

Para abordar este problema, antes de utilizar la función `randperm` de MATLAB que permuta las muestras del primer *dataset* de forma aleatoria, se ha concatenado un vector columna con el *dataset*. Cada elemento de este vector indica la posición original de cada muestra. Una vez realizada esta permutación, se extraen los índices que indican a qué posición se ha ido cada muestra. Estos índices se utilizan para permutar las muestras del segundo *dataset* de la misma manera que se ha hecho con el primero.

4.4 Implementación

Para realizar las pruebas sobre el *dataset* se ha implementado un *script* que sigue los siguientes pasos:

1. Se establece el número total de pruebas a realizar.
2. Se seleccionan las *features* que se tendrán en cuenta.
3. Se cargan los vectores de características de cada clase que se desee tener en cuenta procedentes de la grabadora y de la Raspberry.
4. Permutación de las muestras de cada clase de forma aleatoria en los dos *datasets*.
5. Limitación del número máximo de muestras totales por clases.
6. Creación del *training* y del *test dataset*.
7. Se descartan las *features* que no deseen tener en cuenta.
8. Normalización del *dataset*.
9. Entrenamiento de dos clasificadores SVM, uno con las muestras de la grabadora y otro con las muestras de la Raspberry.
10. Entrenamiento de dos clasificadores *Random Forest*, uno con las muestras de la grabadora y otro con las muestras de la Raspberry.

11. Se predicen con los clasificadores entrenados los resultados que se deberían obtener en el *test dataset* de muestras de la grabadora y de la Raspberry.
12. Se calculan, se muestran y se guardan las matrices de confusión.
13. Se escribe la precisión obtenida en cada clasificador en un archivo de Excel.
14. Si no es la última prueba, se vuelve al paso 3.
15. Fin de la ejecución.

En la Figura 4.5 se observa un diagrama de funcionamiento de la función. Se han utilizado las funciones de MATLAB `templateSVM` y `fitcecoc` para crear y entrenar los clasificadores SVM [21]. Para crear y entrenar los clasificadores *Random Forest* se han utilizado las funciones `templateTree` y `fitcensemble` [22].

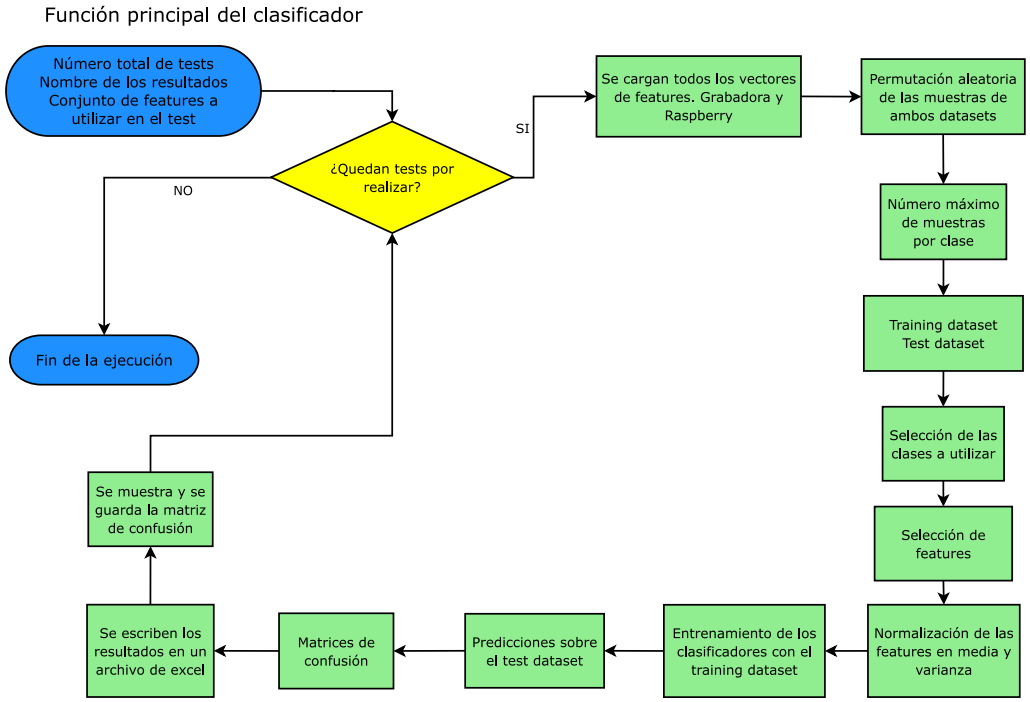


Figura 4.5: Descripción de la función que realiza las pruebas con el clasificador. Fuente: Elaboración propia.

CAPÍTULO 5
Resultados

En este capítulo se comentarán los resultados obtenidos en todas las pruebas de clasificación realizadas en la base de datos de la que se ha partido. Se entiende como *prueba* el proceso en el cual se entrena y se obtiene la precisión resultante de un clasificador. Como se ha observado en la Sección 2.5, se ha partido de una base de datos con 10 clases iniciales. La primera tarea que se plantea es entrenar el clasificador con estas clases y observar el comportamiento que tendrá a la hora de calcular la precisión de cada clase. Para realizar este proceso se han utilizado las *features* obtenidas con las grabaciones de la grabadora, ya que, son las que tienen mejor calidad.

No ha sido nada sencillo decidir el orden de las tareas a realizar para la mejora y optimización del sistema poniéndose en la situación de que se implemente en un dispositivo, en el cual es necesario elegir qué clasificadores se van a utilizar, qué clases se van a distinguir, el tamaño de las tramas de audio, el clasificador que se utilizaría y el conjunto de *features* más óptimo. Finalmente, después de haber realizado algunas pruebas, la metodología que se siguió a la hora de optimizar el modelo fue la que se presenta a continuación:

1. Evaluar y reducir de las clases del *dataset*.
2. Limitar el número de muestras totales a utilizar
3. Elegir el número de *features* más óptimo
4. Elegir el mejor clasificador
5. Elegir el tamaño de trama

Cuando se realiza un *test* sobre el clasificador lo que sucede es que se pasan las muestras destinadas a este propósito por el clasificador entrenado y se comparan las salidas del sistema con las clases a las cuales realmente pertenecen las muestras. Una herramienta conocida como matriz de confusión permite mostrar en una gráfica esta comparación. Esta matriz contiene en sus columnas las clases que predice el clasificador para las muestras de *test* y en sus filas las clases a las que realmente pertenecen estas muestras. En este caso se pondera el resultado en un porcentaje de muestras de cada clase, siendo el número que aparece en cada celda de la matriz el porcentaje de muestras de *test* de una clase. La diagonal de esta matriz son predicciones realizadas de manera correcta y los valores de fuera de diagonal corresponden a los errores de falso rechazo (muestras pertenecientes a una clase determinada rechazadas por el clasificador) y de falsa alarma (muestras que no pertenecen a una clase determinada catalogadas en esa clase). A partir de la diagonal de esta matriz se obtiene la precisión del clasificador por cada clase. La precisión

total del clasificador se obtiene promediando la precisión de cada clase.
Esta matriz se obtiene cuando el clasificador ha observado todas las muestras que estaban en la parte de *test* y las ha clasificado. En la Figura 5.1 se observa un ejemplo de matriz de confusión.

5.1 Evaluación con las muestras tomadas con la grabadora

Para evaluar el comportamiento de las diferentes clases del dataset se han utilizado los clasificadores SVM y *Random Forest* con un tamaño de trama de 1024 muestras y las 301 características que componen las muestras originales tal y como se especifica en la Sección 3.4. Primero se han realizado 5 pruebas de entrenamiento y *test* para evaluar qué clases presentan más estabilidad en las diferentes pruebas realizadas y si existe alguna confusión generalizada de una clase con otra. En la Figura 5.1 se observa la matriz de confusión que se obtiene tras hacer una prueba, tanto para un clasificador SVM como para el *Random Forest*.

Observando las matrices de confusión que se obtienen en cada una de las 5 pruebas se observa que la clase *Traffic+Voices* tiende a ser clasificada como *Traffic*, la clase *Voices+Traffic* como *Voices* y en menor parte como *Traffic*, y que *Unknown* se etiqueta como *Traffic* mayormente. Tras lo comentado, se ha decidido reetiquetar las muestras de *Voices+Traffic* como *Voices*, las de *Traffic+Voices* como *Traffic* y las de la clase *Unknown* como *Traffic*. El script de MATLAB que se ha implementado permite realizar de una forma sencilla este proceso sin necesidad de cambiar las muestras de directorio, simplemente se cambia el valor de salida que se le da.

Dado a que en el *dataset* que se ha utilizado hay una gran descompensación con el número de muestras existentes por cada clase, habiendo por un lado 1367 muestras de características de sonido de tráfico y 15 muestras de características para sonido de sirena por otro lado, se decide limitar el número de muestras a utilizar en las pruebas a un máximo de 200. De estas 200 hay que recordar que el 80 % van destinadas a entrenar los algoritmos de *Machine Learning* y el 20 % para obtener la precisión de cada clasificador. Teniendo en cuenta lo que se ha comentado en el párrafo anterior, se decide en las siguientes pruebas descartar las clases *Unknown*, *Voices+Traffic* y *Traffic+Voices*, utilizando las muestras de *Traffic* sólamente para los conjuntos de vectores de características utilizados para clasificar sonido de tráfico.

En la Figura 5.2 aparece el número máximo de muestras que se utilizarán por cada clase en las siguientes pruebas que se realizarán en los clasificadores.

Con las mismas *features* que en la prueba anterior, se realizan las mismas pruebas que se han realizado anteriormente teniendo en cuenta las modificaciones que se han realizado en el modelo. El hecho de eliminar las clases *Unknown*, *Voices+Traffic* y *Traffic+Voices* y limitar el número máximo de muestras a utilizar a un máximo de 200 ha mejorado considerablemente la precisión del clasificador. Para estas pruebas del modelo se decide también utilizar unos conjuntos de *features* distintos para observar cómo se comporta cada clase según qué características se utilicen. Se opta primero por 4 combinaciones diferentes de *features*:

- Las 301 de las que se han partido.
- Las 163 que se utilizan en [16].
- Las 275 correspondientes a las relacionadas sólamente con los MFCC utilizadas en [4].

- 288 que serían combinación de utilizar las de [16] con las de [4].

En la Figura 5.3 se observa la precisión total obtenida para cada clasificador y para cada set de características y en la Figura 5.4 se observa la precisión obtenida distinguiendo cada clase para cada clasificador y set de *features*. En base a estos resultados se llegan a las siguientes conclusiones con lo que respecta a las clases utilizadas:

- *Siren*: A pesar de conseguir una precisión medianamente razonable en los sets de características que conforman 288 y 275 *features*, la que se obtiene utilizando 163 dista mucho de ser una precisión buena. Esto es debido a que el número de muestras obtenidas para esta clase ha sido muy escaso, siendo en total 15.
- *Claps* y *Whistle*: Teniendo pocas muestras en ambas clases, se consigue que den una precisión de clasificación estable en los distintos conjuntos de características. Esto es debido a que ambos son sonidos impulsivos, estos sonidos son fáciles de reconocer con las *features* que se utilizan en [4]. En el caso de la clase relacionada con los ruidos de silbato, hay que añadir que es un sonido con una frecuencia muy determinada, factor que ayuda a que sea más fácil de distinguir.
- *Birds* y *Horn*: En ambas clases se consigue una precisión inferior al 60 % en todos los conjuntos de *features* y en ambos clasificadores. A pesar de tener más muestras que *Claps* o *Whistle*, tras haber escuchado de nuevo las tomas que forman parte de estas clases, se ha concluido que esta falta de precisión se debe al dominio del ruido de fondo que hay sobre la fuente de sonido y a la vez, por la poca presencia que tienen en algunas tomas.
- *Voices* y *Traffic*: Ambas clases dan muy buenos resultados en la clasificación, debido a que se poseen muchas más muestras de estas clases que del resto y que en las tomas hay una gran presencia de estas fuentes de ruido.

Para seguir con el estudio del *dataset*, se deja de tener en cuenta las clases: *Siren*, *Birds* y *Horn*. Esta simplificación se realiza porque para estas 3 clases no se obtiene una precisión del clasificador óptima, además de que son clases en las cuales las tomas contienen un ruido de fondo elevado (*Birds* y *Horn*) o que el número de muestras que poseen no es lo suficientemente alto como para poder enseñar a un clasificador a que generalice adecuadamente para nuevas observaciones de la clase (como sucede en *Siren*).

5.1.1. Selección de características

Así también, se ha observado si existe alguna dependencia entre los valores de las características utilizadas. Para ello, se han representado los valores de una *feature* y de todas las muestras con respecto a los valores de una *feature* z. Si los valores de z e y muestran un comportamiento lineal, significa que existe una correlación entre ambas y que están aportando la misma información al clasificador teninendo un efecto similar a introducir la misma *feature* dos veces, pudiendo así, dar más peso a esa determinada característica. Otra manera de verificar si existe alguna dependencia entre características es escoger un par de *features* y sobre todas las muestras calcular una recta de regresión. Si las características se ajustan a esta recta de regresión teniendo un error bajo, se puede determinar que existe una dependencia lineal entre este par de características.

En este trabajo, se ha observado que existe una dependencia lineal entre el ZCR y la media del *Spread*, y entre el ZCR y la media del centroide. Se decide elaborar nuevos conjuntos de *features* en los que se deja de tener en cuenta el *Spread* y el centroide para tener en cuenta el ZCR, consiguiendo eliminar 2 características.

También se encontró dependencia entre el mínimo y la media de los primeros coeficientes de los MFCC del segmento, y la mediana y la media de los primeros coeficientes de los MFCC del segmento. Estas dos últimas no se tienen en cuenta, ya que, al eliminar una de característica de estos pares dependientes empeoraban los resultados de clasificación.

En la Figura 5.5 se muestran dos ejemplos en los que se representan los valores de dos pares de *features* de las muestras de 4 clases. En la imagen de la izquierda se observan dos pares de features con baja correlación y a la derecha se muestran dos pares de características con una alta correlación

Con todo lo anterior, para las siguientes pruebas se establece el tamaño de trama a 2048 muestras digitales de audio y el número de pruebas por conjunto de características aumenta a 10. En la Tabla 5.1 se muestran todas las combinaciones de características que se utilizarán para estas pruebas con tal de comprobar qué combinación puede dar mejores resultados y cómo pueden mejorar o empeorar los resultados de la clasificación según se eliminan o se incluyen características. Para referirse a un conjunto concreto de características se utilizarán los nombres de la Tabla 5.1. Los conjuntos de características llamados *161median* y *161minmax* se han elaborado después de estas pruebas al observar que los máximos y mínimos de los MFCC, y las medianas de los MFCC aportaban mejoras en la precisión del clasificador, esto se comentará más tarde.

Tabla 5.1: Todas las combinaciones de *features* utilizadas. Fuente: Elaboración propia.

	161	163	163minmax	163median	163skekur	275	275ZCR	275temp	275spec	288	161median	161minmax
MFCC												
Medias MFCC (25)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Varianzas MFCC (25)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mínimos MFCC (25)			✓			✓	✓	✓	✓	✓		✓
Máximos MFCC (25)			✓			✓	✓	✓	✓	✓		✓
Medianas MFCC (25)				✓		✓	✓	✓	✓	✓	✓	
Skewness MFCC (25)					✓	✓	✓	✓	✓	✓		
Curtosis MFCC (25)					✓	✓	✓	✓	✓	✓		
Medias Δ-MFCC (25)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Varianzas Δ-MFCC (25)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Medias ΔΔ-MFCC (25)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Varianzas ΔΔ-MFCC (25)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Espectrales												
Media Decrease	✓	✓	✓	✓	✓				✓	✓	✓	✓
Media Flux	✓	✓	✓	✓	✓				✓	✓	✓	✓
Varianza Flux	✓	✓	✓	✓	✓				✓	✓	✓	✓
Media Curtosis	✓	✓	✓	✓	✓				✓	✓	✓	✓
Media Roll-Off	✓	✓	✓	✓	✓				✓	✓	✓	✓
Media Skewness	✓	✓	✓	✓	✓				✓	✓	✓	✓
Media Slope	✓	✓	✓	✓	✓				✓	✓	✓	✓
Media Centroide		✓	✓	✓	✓				✓	✓		
Media Spread		✓	✓	✓	✓				✓	✓		
Temporales												
Zero Crossing Rate	✓	✓	✓	✓	✓		✓	✓		✓	✓	✓
Log-Attack	✓	✓	✓	✓	✓			✓		✓	✓	✓
Pitch	✓	✓	✓	✓	✓			✓		✓	✓	✓
Índice de modulación	✓	✓	✓	✓	✓			✓		✓	✓	✓
Número total de features	161	163	213	188	213	275	276	279	284	288	186	211

Tras estas pruebas, para cada clasificador se obtienen las precisiones que se muestran en la Figura 5.6.

Observando la precisión total de cada clasificador, `288` con un SVM da el mejor resultado con un 91.09 % de precisión. Luego se encuentra un *Random Forest* con el conjunto `275spec` con 284 características seguido del mismo conjunto de *features* pero con un clasificador SVM obteniendo un 89.34 % y un 88.31 %.
Pero como se pretende encontrar un conjunto de características más reducido que obtenga una buena precisión, se tiene que con `163median` (que tiene 188 *features*) utilizando un

SVM se obtiene una precisión del 86.94 %. Por otra parte, el conjunto que menos características utiliza es `161` (que contiene 161 *features*) da como resultado una precisión total del 84.68 % utilizando un clasificador SVM.

Teniendo en cuenta los resultados obtenidos por clases:

- *Traffic*: Por lo general esta clase de ruido tiene muy buena precisión (entre el 99.5 % y el 94 % en todos los clasificadores y conjuntos de features utilizados). Esta clase no se podría considerar en este caso la limitante de la precisión total, dando mucho margen para poder elegir clasificadores que si que se adapten mejor al resto de clases.
- *Voices*: Se encuentran valores de precisión entre el 94.25 % y el 86.5 %. El mejor clasificador para esta clase es un SVM que utiliza el conjunto de *features* `161`.
- *Claps*: En esta clase ya se encuentra un rango de precisión más amplio, estando entre 96.36 % y el 70 %. La precisión viene liderada por el conjunto `275spec` con un *Random Forest*. En segunda posición el conjunto `288` con un SVM obteniendo una precisión del 86.36 %. Por último, en la tercera se encuentra `163median` con un 82.72 % utilizando un *Random Forest*.
- *Whistle*: Esta clase es la que mayor rango resulta, estando entre el 86.25 % y el 47.5 % de precisión. Esta es la clase más sensible del sistema. El conjunto `288` es el que da el mejor resultado con un SVM. Seguido se encuentran el conjunto `163minmax` utilizando un *Random Forest* y el `275zcr` con un SVM obteniendo una precisión del 81.25 %.

La precisión que obtiene cada conjunto de *features* por clases utilizando cada clasificador se observa en la Figura 5.7.

Incluir las medianas de los MFCC o los mínimos y los máximos de los MFCC suele mejorar los resultados en la clasificación, también sucede esto al eliminar el centroide y el *Spread*. En base a los resultados obtenidos por clases y a lo obtenido, se decide probar con dos conjuntos de características nuevos:

- `161median`: Utiliza las características de `161` con las medianas de los MFCC.
- `161minmax`: Utiliza las características de `161` con los mínimos y máximos de los MFCC.

Las precisiones totales que se obtienen con los nuevos conjuntos se muestran en la Figura 5.9. Se observa que el mejor clasificador es un *Random Forest* que utiliza el conjunto `161median`, obteniendo una precisión total del 91.58 %. Se decide entonces que este clasificador y este conjunto de características son los más adecuados para realizar la tarea de clasificación. Así mismo, en la Figura 5.8 se muestra una de las matrices de confusión obtenidas en una prueba realizada con este clasificador.

Por último, con lo que respecta al tamaño de la trama, se decide establecer un tamaño de trama de 2048 muestras, correspondiendo a una duración de 42.67 ms. Esta decisión se ha tomado porque es el tamaño de ventana que ha dado mejores resultados.

5.2 Evaluación por correlación cruzada de bases de datos

Una vez se tiene un conocimiento de cómo se comportan los diferentes parámetros que se tienen que determinar a la hora de afrontar un problema de *Machine Learning* y

se ha realizado un estudio de precisión con las muestras de la grabadora, se procede a medir las precisiones que se obtienen utilizando también las muestras de la Raspberry. En este proceso se realizan 3 conjuntos de pruebas similares a las que se han realizado en la Sección 5.1, en las cuales se utilizarán los algoritmos SVM y *Random Forest* y los diferentes conjuntos de características. En este trabajo se han realizado un total de 4 conjuntos de pruebas, en cada uno de estos conjuntos de pruebas se utilizan un conjunto de muestras de entrenamiento y de *test* procedentes o del mismo *dataset* o de otro distinto, quedando las siguientes combinaciones:

- Entrenamiento: Grabadora. *Test*: Grabadora
- Entrenamiento: Raspberry. *Test*: Raspberry
- Entrenamiento: Grabadora. *Test*: Raspberry
- Entrenamiento: Raspberry. *Test*: Grabadora

Tras todo esto, las precisiones obtenidas para cada clasificador y conjunto de características se muestran ordenadas de mayor a menor precisión en la Tabla 5.2.

Tabla 5.2: Resultados con distintos *datasets* ordenados de mayor a menor precisión. Fuente: Elaboración propia.

Entrenamiento: Grabadora			Entrenamiento: Raspberry			Entrenamiento: Grabadora			Entrenamiento: Raspberry		
Test: Grabadora			Test: Raspberry			Test: Raspberry			Test: Grabadora		
161median	RF	91,58	161median	RF	90,85	288	SVM	92,10	275spec	RF	88,72
288	SVM	91,09	288	SVM	90,82	275temp	SVM	89,70	288	SVM	88,51
275spec	RF	89,34	288	RF	89,24	161median	SVM	89,14	288	RF	87,14
161minmax	SVM	88,56	163minmax	SVM	88,99	288	RF	88,99	161minmax	SVM	87,07
275spec	SVM	88,31	275spec	RF	88,80	275zcr	SVM	88,75	163minmax	SVM	85,64
275temp	SVM	87,11	161minmax	SVM	88,35	275spec	SVM	88,36	275spec	SVM	85,38
163minmax	RF	87,09	275	SVM	88,00	163minmax	SVM	88,29	275	SVM	85,16
163median	SVM	86,94	275spec	SVM	87,80	275spec	RF	88,11	275temp	SVM	84,78
275	SVM	86,85	275temp	SVM	87,79	163median	SVM	88,05	161median	SVM	84,16
163minmax	SVM	86,77	163median	SVM	87,23	161minmax	SVM	87,36	163	SVM	83,68
288	RF	86,53	275zcr	SVM	86,78	275	SVM	86,96	163median	SVM	83,53
161median	SVM	85,95	161median	SVM	86,67	161median	RF	86,79	161	SVM	83,03
275zcr	SVM	85,68	161minmax	RF	85,73	163skekur	SVM	86,64	163median	RF	81,88
163median	RF	84,68	161	SVM	85,53	163	SVM	85,84	163skekur	SVM	81,69
161	SVM	84,68	163skekur	SVM	85,49	161	SVM	84,59	275zcr	SVM	81,66
163	SVM	83,83	163	SVM	85,23	275	RF	83,57	275temp	RF	81,60
163skekur	SVM	83,70	275temp	RF	84,04	163minmax	RF	83,38	163minmax	RF	81,19
161	RF	82,92	275	RF	83,81	275temp	RF	81,59	161	RF	80,25
275	RF	82,14	163	RF	83,78	163median	RF	80,99	275	RF	79,64
275temp	RF	82,09	161	RF	83,41	163	RF	80,96	161median	RF	79,57
161minmax	RF	81,37	163minmax	RF	83,41	161	RF	80,26	275zcr	RF	78,85
163	RF	79,69	163median	RF	83,16	163skekur	RF	80,09	163	RF	78,77
275zcr	RF	79,33	163skekur	RF	82,05	161minmax	RF	78,93	161minmax	RF	76,87
163skekur	RF	76,45	275zcr	RF	81,62	275zcr	RF	77,89	163skekur	RF	75,74

En este trabajo es interesante observar los resultados que se obtienen cuando se entrena utilizando muestras de la grabadora y se realiza el *test* con las muestras de la Raspberry y al revés. Esto es importante para determinar si la calidad del micrófono que se ha utilizado para crear el *dataset* de entrenamiento va a influir en la precisión de un sistema de clasificación implementado con otro micrófono.

Teniendo en cuenta el conjunto de características elegido `161median` que utiliza un *Random Forest*, se tiene que en caso de utilizar las muestras de la Raspberry tanto en entrenamiento y *test* pierde un 0.73 puntos porcentuales en la precisión total, en el caso de utilizar las muestras de la grabadora como entrenamiento y las de la Raspberry como *test*, se pierden 4.79 puntos porcentuales, quedando así una precisión total del 86.79 %. Aunque se pierde bastante precisión, sigue estando en un umbral bastante aceptable. Hay casos en los que utilizando las muestras de la Raspberry en el entrenamiento, indistintamente de las que se utilicen en el *test*, el sistema consigue una precisión mayor que sólo

utilizando muestras de la grabadora. Esto podría ser debido a que el micrófono utilizado en la Raspberry capta más ruido que el de la grabadora.

Comentando los resultados obtenidos entrenando con muestras de la grabadora y realizando el *test* con las muestras de la Raspberry, se debe de tener en cuenta el criterio de utilizar una menor cantidad de *features* posibles. Por lo tanto, los mejores clasificadores dentro de esta categoría serían:

1. `161median` con SVM. Precisión: 89.14 %.
2. `163minmax` con SVM. Precisión: 88.29 %.
3. `163median` con SVM. Precisión: 88.05 %.
4. `161minmax` con SVM. Precisión: 87.36 %.
5. `161median` con *Random Forest*. Precisión: 86.79 %.

Teniendo en cuenta esta lista y la Tabla 5.2, se observa que el clasificador SVM utilizado es el más óptimo cuando se ha utilizado un micrófono de mayor calidad en el entrenamiento con respecto al que se utiliza en el *test*.

Por otra parte, en lo que respecta a los resultados obtenidos entrenando con muestras de la Raspberry y realizando el *test* con las muestras de la grabadora y, teniendo en cuenta el criterio de utilizar el menor número de *features* posibles, se tiene que los mejores clasificadores dentro de esta categoría son:

1. `161minmax` con SVM. Precisión: 87.07 %.
2. `163minmax` con SVM. Precisión: 85.64 %.
3. `161median` con SVM. Precisión: 84.16 %.
4. `163` con SVM. Precisión: 83.68 %.
5. `163median` con SVM. Precisión: 83.53 %.

Igual que antes, el clasificador SVM es el que da mejores resultados. En diferencia a lo anterior, los resultados obtenidos en este caso son peores, esto sería debido a que la calidad del micrófono utilizado en el entrenamiento es inferior al que se utiliza en el *test*.

Encarando estos resultados a una implementación de un sistema real, utilizar un clasificador SVM con un set de *features* como el de `161median` puede dar muy buenos resultados. Sería recomendable que las muestras que se grabasen para el entrenamiento de este clasificador fuese con un micrófono de buena calidad, de esta manera, las pérdidas de precisión producidas por utilizar en el sistema un micrófono de una calidad inferior serán menores.

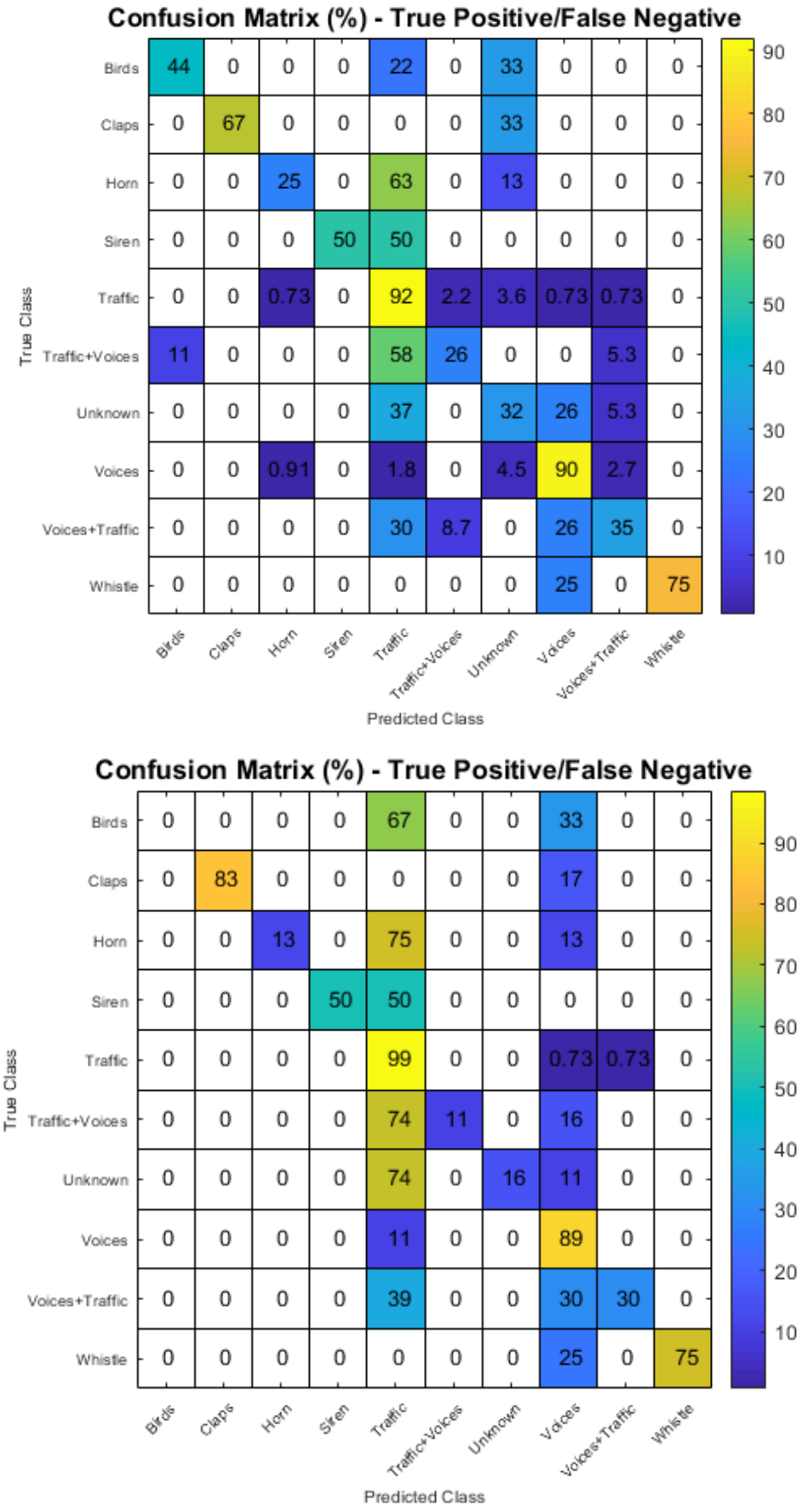


Figura 5.1: Resultados tras la prueba con el clasificador SVM (arriba) y el clasificador *Random Forest* (abajo). Fuente: Elaboración propia.

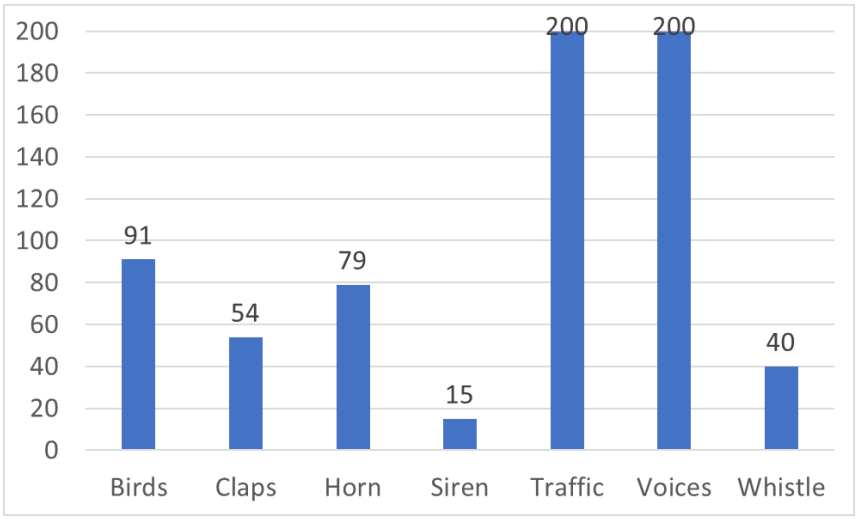


Figura 5.2: Número de muestras limitado a un máximo de 200. Fuente: Elaboración propia.

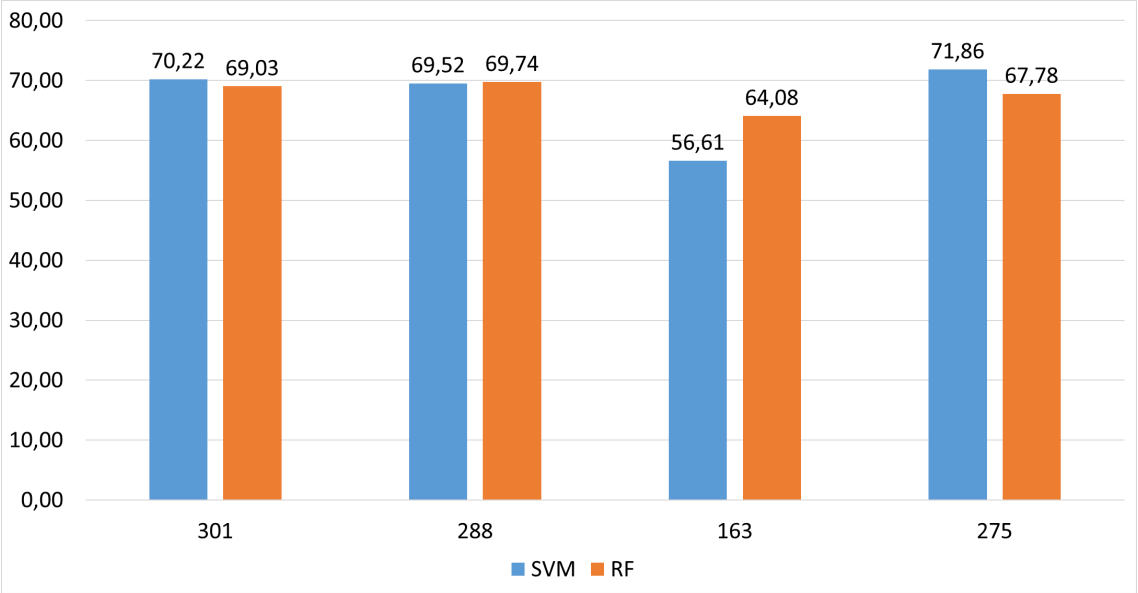


Figura 5.3: Resultados de precisión total tras limitar las muestras. El eje horizontal representa el número de *features* que se han utilizado y el vertical la precisión total de cada clasificador. Fuente: Elaboración propia.

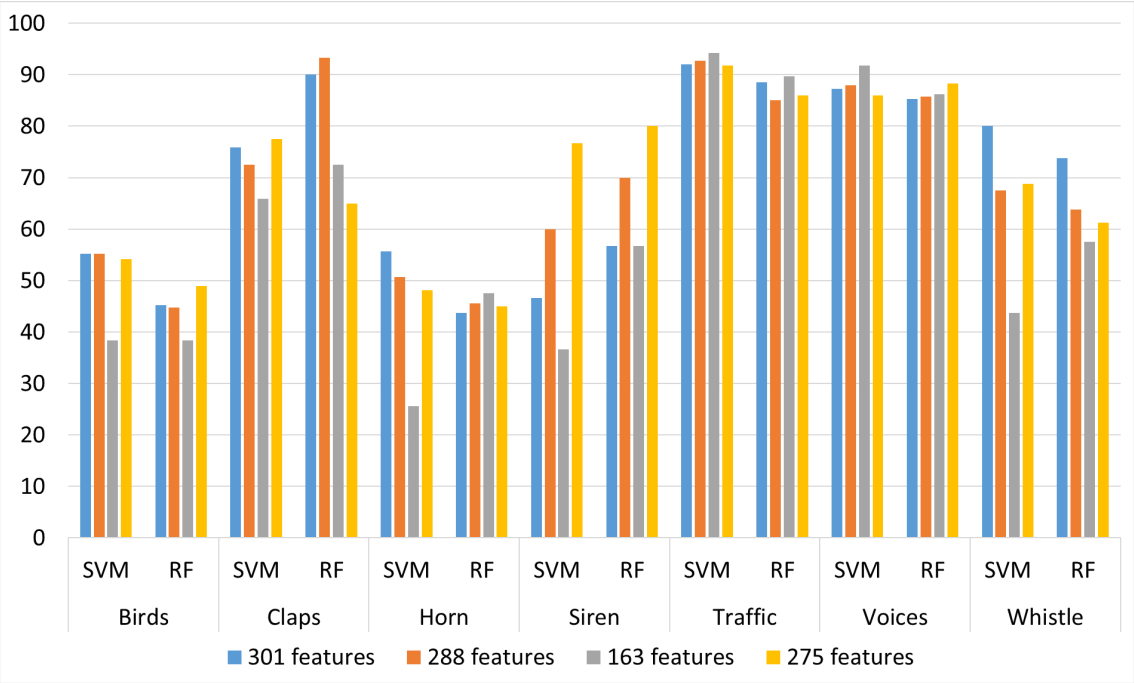


Figura 5.4: Resultados de precisión por clase según el clasificador y el set de características utilizados. Fuente: Elaboración propia.

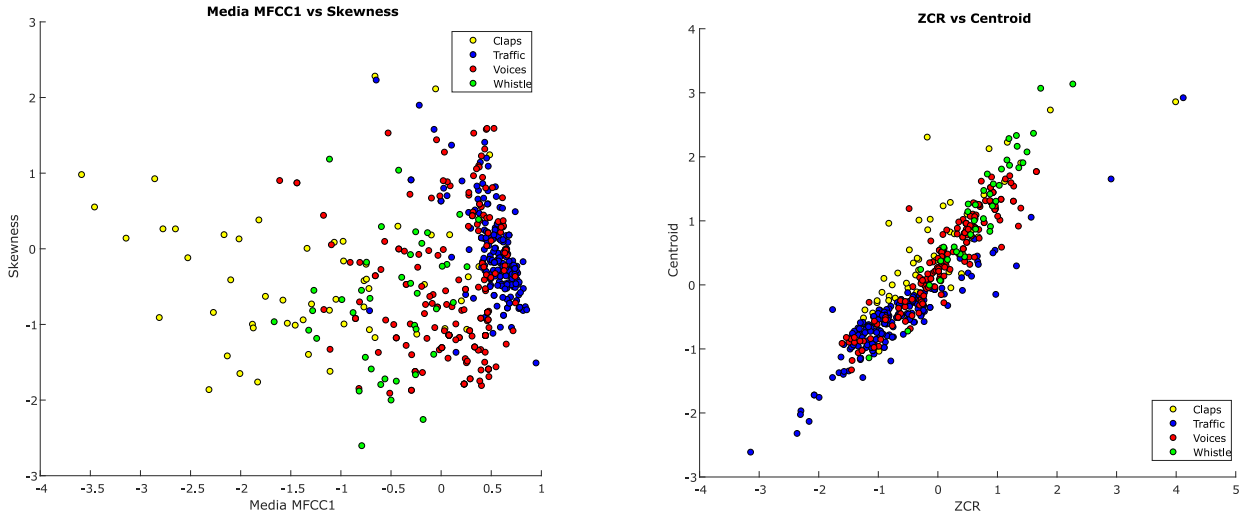


Figura 5.5: Ejemplo de pares de valores de *features*. Fuente: Elaboración propia.

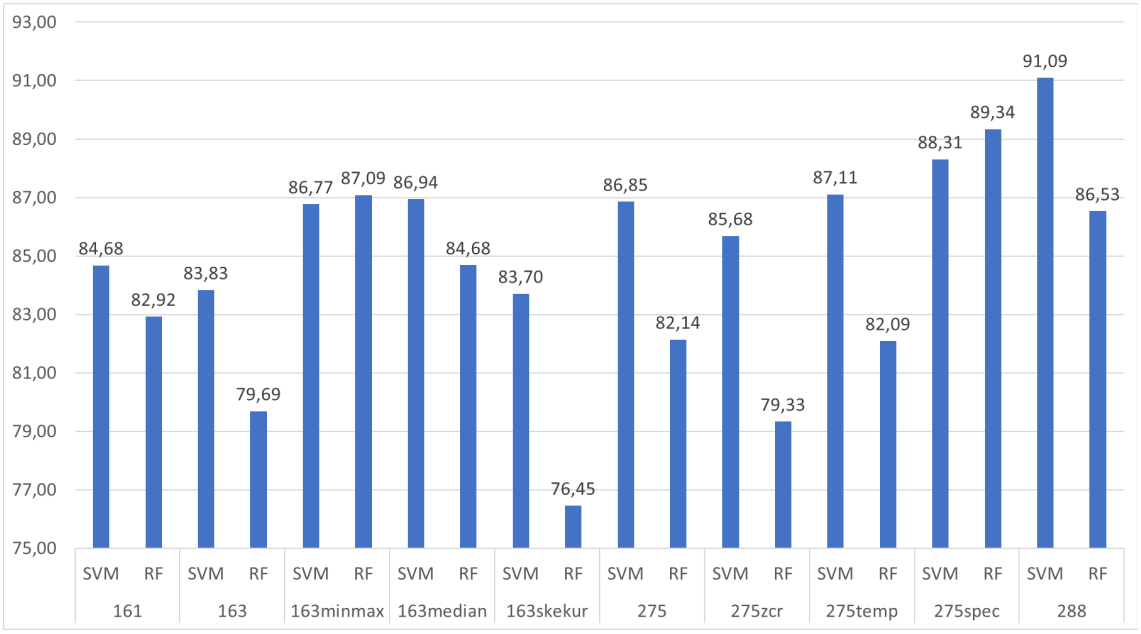


Figura 5.6: Precisión total de cada clasificador según las *features* utilizadas. Fuente: Elaboración propia.

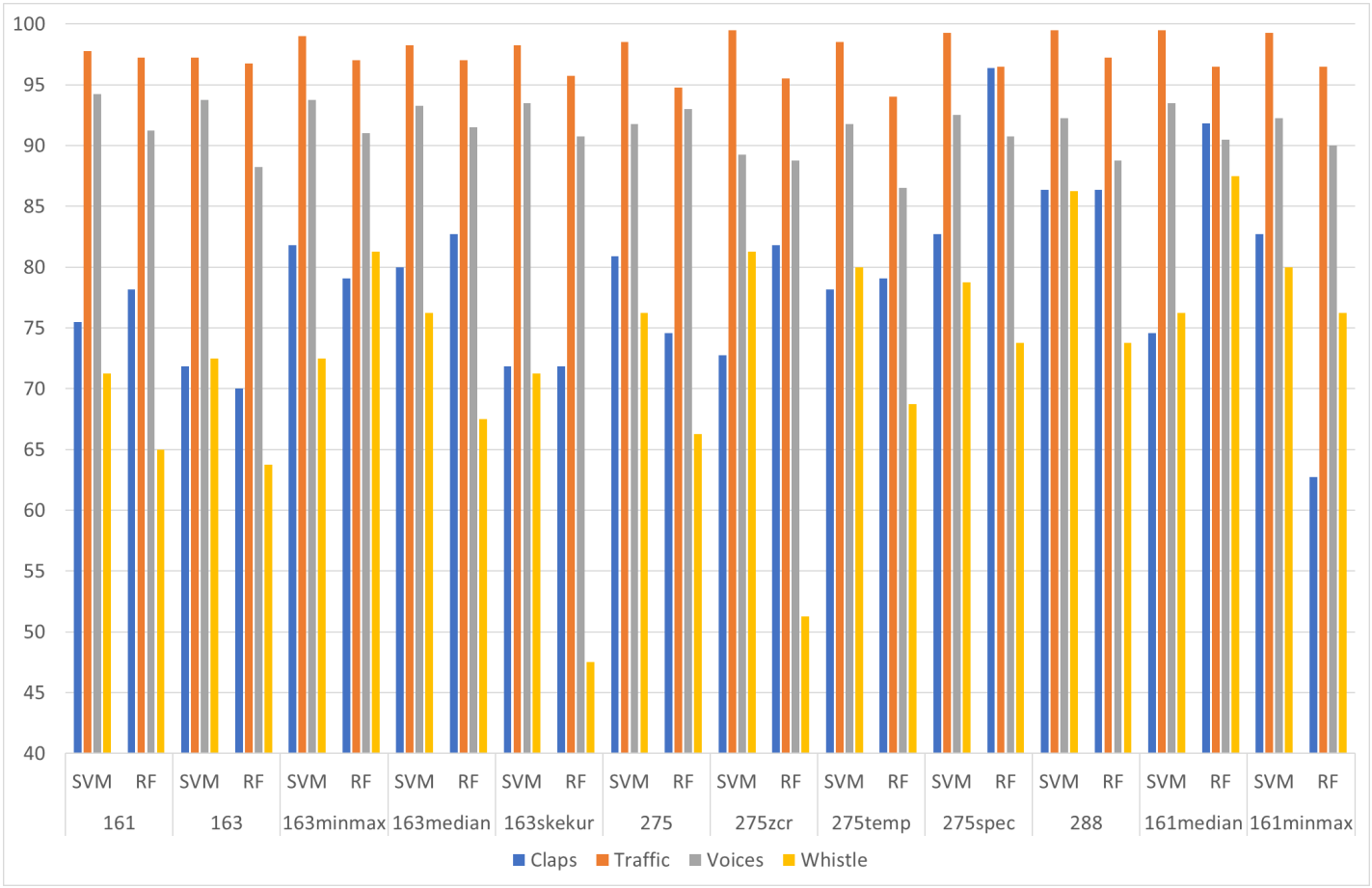


Figura 5.7: Resultados de precisión por clases utilizando distintos conjuntos de *features*. Fuente: Elaboración propia.

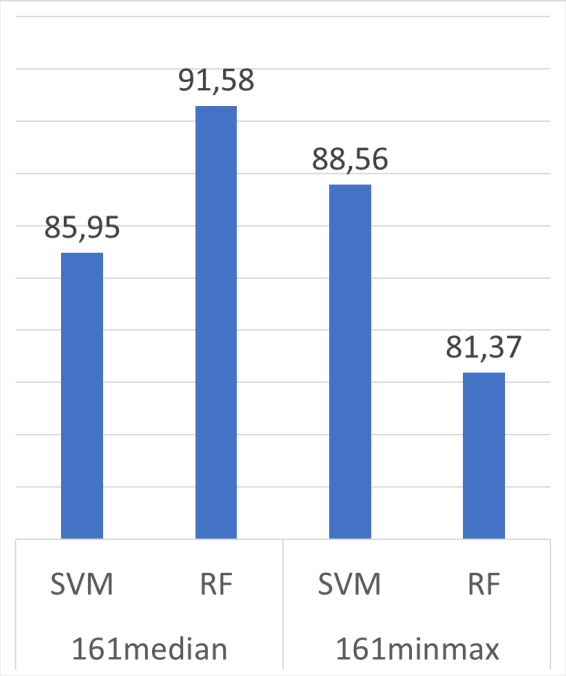


Figura 5.8: Precisión total con `161median` y `161minmax`. Fuente: Elaboración propia.

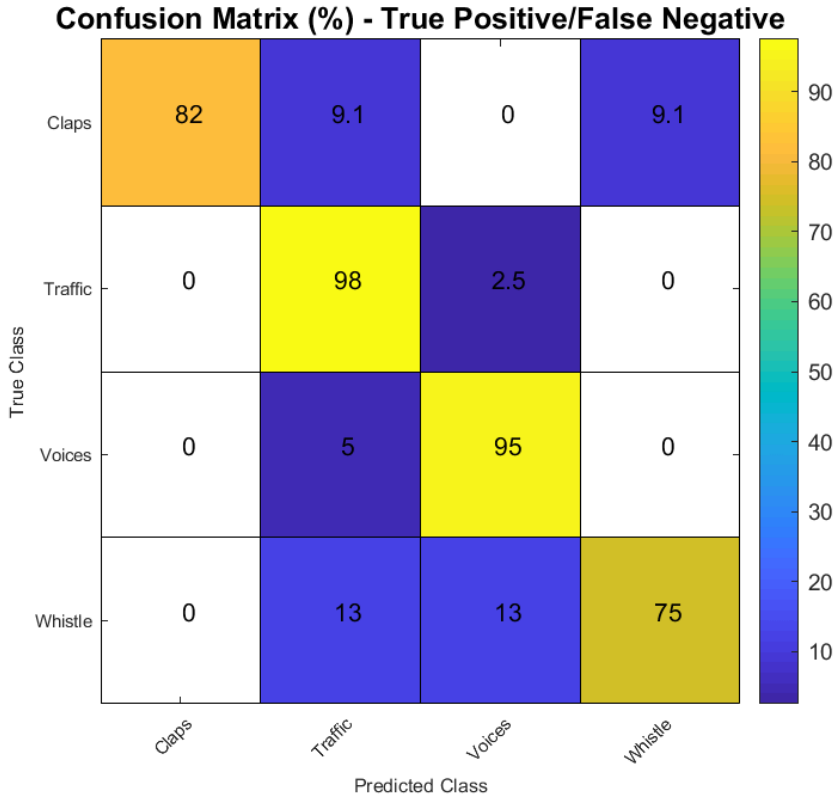


Figura 5.9: Matriz de confusión utilizando `161median` con un *Random Forest* obtenida en una de las pruebas. Fuente: Elaboración propia.

CAPÍTULO 6
Conclusiones

El principal objetivo de este trabajo ha sido implementar un sistema de clasificación de sonidos urbanos de bajo coste. En concreto se ha analizado la influencia de la calidad del sistema en las prestaciones del clasificador utilizado. Del mismo modo, también se ha estudiado una metodología a seguir para abordar este problema utilizando *Machine Learning*.
Se ha logrado crear dos bases de datos con sonidos urbanos etiquetados según la fuente de ruido predominante en cada toma de audio. Las dos bases de datos corresponden, cada una, a dos dispotivos de grabación diferentes. Luego se ha implementado con MATLAB un algoritmo que permite extraer las características que describen los distintos segmentos de audio así como la implementación de una herramienta que permita entrenar distintos tipos de clasificadores puediendo elegir en cada momento las características que se tendrán en cuenta en la tarea de clasificación. Se han evaluado las prestaciones de cada dispositivo de grabación configurando adecuadamente la herramienta implementada.

Las limitaciones de este estudio han sido las escasas muestras que se han obtenido de ruidos de sirenas y la cantidad de ruido de fondo que aparece en las clases de claxon y de pájaros. Por otra parte, también la baja presencia de muestras de sonidos de silbatos y palmadas ha hecho que la precisión que se obtiene distinguiendo estas clases sea inferior en comparación a clases con más muestras como las de tráfico y voz. Cabe añadir que en la tarea de etiquetado de las tomas de audio, ha sido complicado determinar en ciertos puntos el umbral de decisión para catalogar una toma dentro de una clase u otra.

Los resultados que se obtienen revelan que contando con una captación de sonido de menor calidad, los resultados pueden verse afectados unos puntos porcentuales por debajo con respecto a una captación de sonido de mayor calidad. Teniendo en cuenta que se debe de utilizar una cantidad pequeña de características, sistemas que utilizan 161, 163, 186, 188, 211 y 213 *features* han obtenido precisiones superiores al 84 %.
Otra observación que se realiza en base a los resultados es que las tomas más ruidosas tienen tendencia a clasificarse como ruido de tráfico. Esto es debido a que el espectro en frecuencia del ruido de tráfico presenta un espectro en frecuencia muy similar al del ruido. Por lo tanto, es importante que el ruido introducido por el sistema de captación no sea alto.

CAPÍTULO 7
Líneas futuras

Este trabajo abre las puertas a introducir mejoras en el sistema de clasificación y en la metodología que se plantea, así como poder darle una aplicación útil:

Mejoras en el sistema de clasificación:

- Implementación de una herramienta eficiente de segmentación y etiquetado. La fase de segmentación y etiquetado es la que ha consumido más tiempo en todo el proceso, por ello, es necesaria la implementación de una herramienta que permita segmentar y etiquetar grabaciones de audio de una manera más eficiente. De esta manera se ahorrará tiempo, se obtendrá una cantidad de segmentos por minuto mayor y se reducirá el coste económico que podría implicar para una organización la obtención de muestras.
- Obtener muestras de distintas ubicaciones. Como los *datasets* obtenidos se han grabado cerca de avenidas, sería interesante tener en cuenta otros entornos urbanos como calles o incluso plazas en las cuales no hay tanta presencia de tráfico. Esto aportará una mayor diversidad al conjunto de datos.
- Ampliación del *dataset*. Otra mejora a realizar sería la de ampliar la base de datos para obtener un mayor número de muestras de clases con menos presencia y aumentar la calidad de las muestras que se han obtenido para estas clases.
- Optimizar las *features* que se utilizan. Se podría realizar un estudio más exhaustivo para mejorar el proceso de selección de características.

Aplicaciones:

- Implementación de un sistema que permita recopilar a tiempo real las principales fuentes de contaminación acústica en las distintas zonas del entorno urbano pudiendo conocer qué ruidos afectan más en las distintas zonas.
- Dispositivo que ayude a personas con discapacidad auditiva. Se podría desarrollar un dispositivo de bajo coste capaz de ayudar a personas con discapacidad auditiva a reconocer los eventos acústicos que suceden a su alrededor como podrían ser señales acústicas dirigidas hacia estas personas u otros eventos. Un sistema similar se plantea en [23], en el cual es sistema ayuda a enfatizar sonidos en un sistema de audífonos.
- Ayuda de conducción para personas con discapacidad auditiva. Una mejora de este sistema puede permitir implementar un dispositivo que, en un vehículo, permita a una persona con discapacidad auditiva poder visualizar las señales acústicas que

se emiten en el medio, de esta manera, esta persona podrá conocer si alguien ha utilizado un claxon o si se escucha una sirena a lo lejos.

- Automatización de estudios de acústica ambiental. En estos estudios se mide con un sonómetro el nivel de presión sonora de una determinada zona, con la necesidad de anotar en qué instantes suceden determinados eventos acústicos que pueden condicionar el resultado final. Un clasificador de eventos acústicos puede facilitar esta tarea y, en el mejor de los casos, automatizarla.
- Accesorio para radares acústicos. Los radares acústicos disponen de un conjunto de micrófonos que miden el nivel de presión sonora que se está produciendo, que a su vez, realizan una estimación bastante acertada de la posición de la fuente de ruido. Una herramienta de clasificación de sonidos permitiría identificar, sin necesidad de escuchar una grabación, qué clase de ruido ha sido el causante de la infracción, aportando dicha información en tiempo real. La organización que está implementando estos radares es *Bruitparif* [27].

Bibliografía

[1] Material del curso de *Machine Learning* ofrecido por la Universidad de Stanford en la plataforma *coursera*. Disponible: https://www.coursera.org/learn/machine-learning/. [Consultado: 29 de julio de 2019]

[2] T. Giannakopoulos and A. Pikrakis *Introduction to Audio Analysis: A MATLAB Approach*. Elsevier, First edition, 2014.

[3] F. Rong Audio Classification Method Based on Machine Learning. *2016 International Conference on Intelligent Transportation, Big Data & Smart City*, Dic. 2016.

[4] J. Salamon, C. Jacoby and J. P. Bello Dataset and Taxonomy for Urban Sound Research. *22nd ACM International Conference on Multimedia*, Orlando USA, Nov. 2014.

[5] R. Fernandes de Mello and M. Antonelli Ponti *Machine Learning: Practical Approach on the Statistical Learning Theory*. Springer, First edition, 2018.

[6] *Google Maps*. Disponible: https://www.google.es/maps/. [Consultado: 17 de junio de 2019]

[7] Página principal de la grabadora . Disponible: https://zoom-na.com/zoom-h4n-pro-handy-recorder. [Consultado: 3 de julio de 2019]

[8] *Raspbian*. Página principal del sistema operativo *Raspbian*. Disponible: https://www.raspbian.org/. [Consultado: 1 de junio de 2019]

[9] Página donde se describen las características de la Raspberry. Disponible: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Consultado: 30 de mayo de 2019]

[10] Librería *PyAudio*. Disponible: https://pypi.org/project/PyAudio/. [Consultado: 23 de junio de 2019]

[11] Página oficial de SSH. Disponible: https://www.ssh.com/ssh/. [Consultado: 24 de junio de 2019]

[12] *PuTTY*. Página donde se describe y se puede descargar el programa. Disponible: https://www.putty.org/. [Consultado: 25 de junio de 2019]

[13] *FileZilla*. Página principal del *software FileZilla*. Disponible: https://filezilla-project.org/. [Consultado: 3 de julio de 2019]

[14] *Reaper*. Página principal del *DAW* utilizado en el proyecto. Disponible: https://www.reaper.fm/. [Consultado: 12 de marzo de 2019]

[15] B. W. Schuller *Intelligent Audio Analysis*. Springer, First edition, 2013.

[16] F. Aguirre Desarrollo y análisis de clasificadores de señales de audio. *Tesis de Máster*, Gandia, Jul. 2017.

[17] *Audio Toolbox de MATLAB* Disponible: https://www.mathworks.com/help/audio/index.html. [Consultado: 31 de julio de 2019]

[18] Extracción de las *features* espectrales y cepstrales. Diagrama. Disponible: https://www.mathworks.com/help/audio/ref/audiofeatureextractor.html [Consultado: 2 de agosto de 2019]

[19] *Spectral Descriptors del Audio Toolbox de MATLAB* Disponible: https://www.mathworks.com/help/audio/ug/spectral-descriptors.html. [Consultado: 31 de julio de 2019]

[20] Función de MATLAB para extaer los MFCC. Disponible: https://www.mathworks.com/help/audio/ref/mfcc.html. [Consultado: 1 de agosto de 2019]

[21] Funciones de MATLAB para entrenar un clasificador basado en SVM Disponible en los dos siguientes enlaces: https://www.mathworks.com/help/stats/fitcecoc.html. https://www.mathworks.com/help/stats/templatesvm.html. [Consultado: 2 de agosto de 2019]

[22] Funciones de MATLAB para entrenar un clasificador basado en *Random Forest* Disponible: https://www.mathworks.com/help/stats/fitcensemble.html. https://www.mathworks.com/help/stats/templatetree.html. [Consultado: 5 de agosto de 2019]

[23] R. Gil-Pita, H. Sanchez-Hevia, C. Llerena-Aguilar, I. Mohino-Herranz, M. Utrilla-Manso and M. Rosa-Zurera Distributed and collaborative sound environment information extraction in binaural hearing aids. *2016 IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM)*, Rio de Janerio, 2016, pp. 1-5.

[24] Página principal del GTAC Disponible: https://gtac.webs.upv.es/. [Consultado: 23 de junio 2019]

[25] Página donde se ha extraído la *datasheet* del UMIK-1. Disponible: https://www.minidsp.com/images/documents/Product%20Brief%20-%20Umik.pdf. [Consultado: 3 de julio de 2019

[26] J. Abeßer, R. Gräfe, C. Kühn, T. Claß, H. Lukashevich, M. Götze, S. Kühnlenz A Distributed Sensor Network for Monitoring Noise Level and Noise Sources in Urban Environments *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, Barcelona, 2018.

[27] Página principal de *Bruitparif* Disponible: https://www.bruitparif.fr/. [Consultado: 7 de septiembre de 2019]

APÉNDICE A
Códigos utilizados

A.1 Código en Python utilizado para grabar con la Raspberry y el micrófono USB

Código desarrollado en el GTAC [24] del iTeam de la UPV.

```
import pyaudio
import wave

# Sample rate: 48 kHz. Resolution: 16 bits. 1 channel
chunk = 960
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 48000
# 7 wav files of  ten minutes recording audio
RECORD_SECONDS = 600
num_max = 7
times = 0

while times < num_max:
    p = pyaudio.PyAudio()

    stream = p.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=RATE,
                    input=True,
                    output=True,
                    frames_per_buffer=chunk)

    print("* recording")
    frames = []
    print("times = %i" %times)
    for i in range(0, int(RATE / chunk * RECORD_SECONDS)):
        data = stream.read(chunk)
        frames.append(data)

    print("finished recording")

    file_name_with_extension = "date_4_7_2019_time_" + str(times) + ".wav"

    stream.stop_stream()
    stream.close()
    p.terminate()

    wf = wave.open("wavs/" + file_name_with_extension, 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(2)
```

```
42      wf.setframerate(RATE)
43      wf.writeframes(b''.join(frames))
44      wf.close()
45
46      print("Saved wav file: %s" %file_name_with_extension)
47      times += 1
```

APÉNDICE B

Datasheet del micrófono *UMIK-1*.

Extraído de la página oficial de miniDSP [25].

miniDSP

UMIK-1

The UMIK-1 is an omni-directional USB measurement microphone providing Plug & Play acoustic measurement. From speaker & room acoustic measurement to recording, this microphone provides low noise and accurate results you can rely on. Forget about driver installation, OS compatibility and un-calibrated mics. The Umik-1 is a USB Audio class 1 device automatically recognized by all Operating Systems (Windows/Mac/Linux). It is provided with a unique calibration file based on the serial number. Time to finally focus on your measurements with three simple steps: Unpack, Plug, Measure!

Technical specifications

Item	Description
Capsule Type & Polar pattern	6mm electret, Omni-Directional
Microphone body	Die cast aluminum
Frequency response	20 Hz - 20kHz +/-1dB with calibration loaded
USB Audio	USB Audio class 1.0 Driverless interface for Windows, Mac & Linux
Resolution & Sample rate	24bit ADC @ 48kHz
Max SPL for 1% THD @ 1kHz	133dB SPL @ 0dB analog gain setting
Calibration file	Unique microphone calibration .txt file referenced to the Serial Number Includes on axis & 90deg - Frequency / Amplitude / Sensitivity drift
Power	USB Powered (5V) - Blue LED indicates unit under Power
Weight	120gm for microphone, 600gm complete kit with accessories
Connector	miniUSB connector
Accessories	1 x hard cardboard case 1 x 2m long USB cable 1 x pivotal microphone mini-stand 1 x microphone clamp 1 x wind screen

Template applications

- Speaker development
- Room acoustic measurements
- Live Sound tuning
- High quality recording

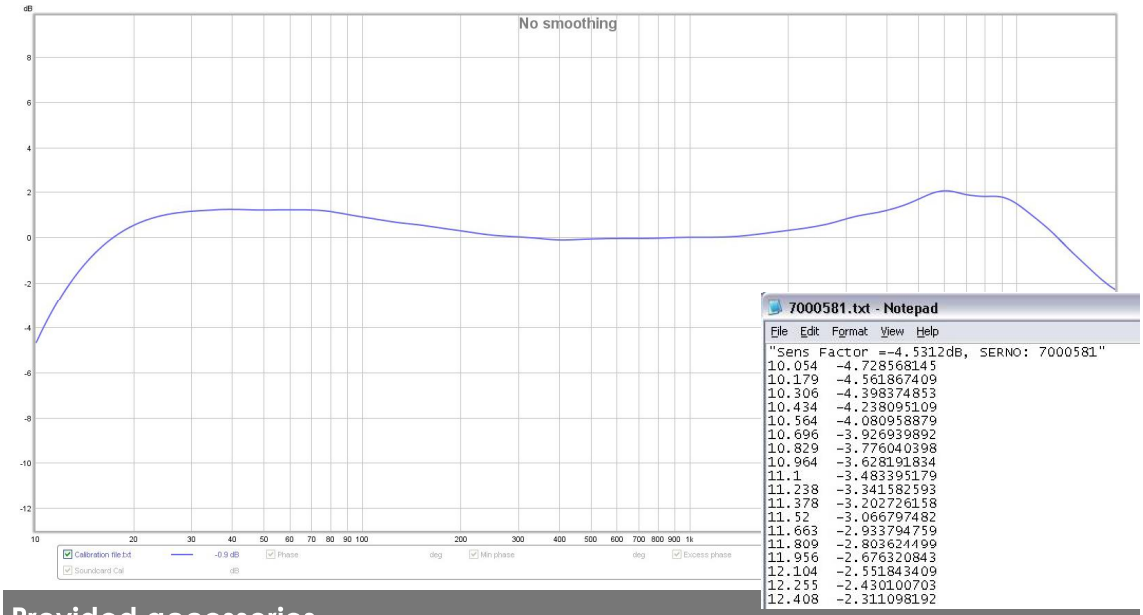
Compatibility

- Windows/Mac/Linux
- All measurement softwares
- Special features such as calibrated SPL measurement enabled under free Room EQ Wizard (REW) and Dirac Live.

miniDSP Ltd - Features and Specifications are subject to change without prior notice
www.miniDSP.com

UMIK-1

Template calibration file loaded into Room EQ Wizard



Provided accessories