ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Desarrollo de un Detector de Eventos Sonoros en Python

AUTOR: Jorge García-Arcicollar Ramírez

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR: Rubén Fraile Muñoz

DEPARTAMENTO: Ingeniería Telemática y Electrónica

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Francisco Javier del Río Martín

TUTOR: Rubén Fraile Muñoz

SECRETARIO: Elena Blanco Martín

Fecha de lectura: 25 de Enero de 2021

Calificación:

El Secretario,

RESUMEN

El espectro de modulación de la envolvente (EMS) es un análisis espectral de las modulaciones de amplitud (para frecuencias bajas) de la envolvente del sonido, dentro de bandas de frecuencia específicas. Fue estudiado en trabajos científicos relacionados con la inteligibilidad de la voz para diferenciar tipos de disartria en el habla.

El EMS puede ser utilizado para caracterizar un conjunto de señales de audio, de modo que esa caracterización puede servir para clasificar una señal de audio dentro de ese grupo de señales conocidas.

El objetivo principal de este proyecto es el desarrollo de un sistema que recibe como entrada una señal de audio, y puede clasificar esta señal dentro de un conjunto de señales conocidas, mediante el uso del EMS para la caracterización de las señales, identificando también el intervalo de tiempo en el que se haya producido. Esto permite el reconocimiento de distintos tipos de sonidos.

La clasificación se realiza tanto para señales provenientes de una única fuente sonora como para señales que contienen fuentes de sonido solapadas en el tiempo, en un entorno ruidoso.

La asociación entre la nueva señal recibida y el grupo de señales ya caracterizadas es realizada mediante el uso de librerías de software que utilizan técnicas de inteligencia artificial, tales como redes neuronales, o descomposición basada en factorización de matrices no negativas.

Para la evaluación del rendimiento del sistema se han utilizado las fuentes de sonido y los criterios de evaluación de la tarea nº 2 del reto DCASE 2016, llamada *Sound event detection in synthetic audio* (detección de eventos sonoros en audio sintetizado), donde se propone un sistema entrenado con diversos sonidos. Además se utiliza el sistema para la detección y clasificación de sonidos mediante la captura de señales en tiempo real.

El análisis de los resultados muestra que el desarrollo en Python de algoritmos que por un lado caracterizan la señal mediante el uso del EMS, y por otro la clasifican utilizando librerías de aprendizaje automático, es adecuado para la detección de eventos de sonido, tanto sobre archivos grabados como en la captura de señales de audio en tiempo real, si bien, en este caso, con un cierto retardo en la detección.

ABSTRACT

The Envelope Modulation Spectrum (EMS) is a spectral analysis of the amplitude modulations (for low frequencies) of the sound envelope, within specific frequency bands. It was studied in scientific works related to the intelligibility of the voice to differentiate types of speech dysarthria.

EMS can be used to characterize a set of audio signals, so that characterization can be used to classify an audio signal within that group of known signals.

The main objective of this project is the development of a system that receives an audio signal as input, and can classify this signal into a set of known signals, by using EMS to characterize the signals, identifying as well the time period in which it occurred. This allows us to recognize different types of sounds.

The classification is made both for signals from a single sound source and for signals that contain time-overlapping sound sources in a noisy environment.

The association between the new received signal and the group of already characterized signals is made through the use of software libraries that use artificial intelligence techniques, such as neural networks, or decomposition based in non negative matrix factorization.

For the evaluation of the performance of the system, the sound sources and the evaluation criteria of the task number 2 of the DCASE 2016 Challenge, called Sound event detection in synthetic audio, have been used, where it is proposed a trained system with various sounds. In addition, the system is used to detect and classify sounds by capturing signals in real time.

The analysis of the results shows that the development in Python of algorithms that, on the one hand, characterize the signal by means of the use of EMS, and on the other classify it using machine learning libraries, is suitable for the detection of sound events, both on recorded files as in the capture of audio signals in real time, although, in this case, with a certain delay in the detection.

Tabla de Contenido

LISTA DE ACRÓNIMOS 8

1. INTRODUCCIÓN 9

2. MARCO TECNOLÓGICO 11

Extracción de características 11

Clasificación 11

3. ESPECIFICACIONES Y RESTRICCIONES DE DISEÑO 13

3.1 IDE: Pycharm 13

3.2 Lenguaje de Programación Python 15

4. DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA 17

4.1 EXTRACCIÓN DE CARACTERÍSTICAS DE SEÑAL: EMS 17

4.2 FILTROS GAMMATONE 19

4.3 MODELO DE CLASIFICACIÓN 1: NMF 21

4.4 MODELO DE CLASIFICACIÓN 2: REDES NEURONALES 22

4.5 ARQUITECTURA DEL SISTEMA 24

4.5.1 Entrenamiento 25

4.5.2 Entrenamiento con ruido añadido 26

4.5.3 Entrenamiento para la detección en tiempo real 29

4.5.4. Detección NMF 30

4.5.5. Detección Neural Network 31

4.5.6. Detección Tiempo Real 32

5. RESULTADOS 37

5.1 EVALUACION DE RESULTADOS 37

5.2 PRUEBAS REALIZADAS 39

5.2.1 Algoritmo de clasificación NMF 39

5.2.2 Algoritmo de clasificación MLP (Neural Networks) 41

5.2.3 Comparación Rendimiento Python-Matlab para cálculo del EMS 44

5.2.4 Detección en Tiempo real 45

6. CONCLUSIONES 46

7. REFERENCIAS 48

8. ANEXOS 52

ANEXO I: PRESUPUESTO 52

ANEXO II: MANUAL DE USUARIO 53
2.1 Instalación 53
2.2 Librerías importadas 53
2.3 Descripción general de módulos Python creados 54
2.4 Detección con NMF 55
2.5 Detección con modelo de Neural Networks 56
2.6 Detección en tiempo real 58

Figuras

Figura 1. Interfaz principal de Pycharm 14
Figura 2. Listado de librerías Python utilizadas 16
Figura 3. Diagrama de bloques del EMS 17
Figura 4. Respuesta en Frecuencia de Banco de Filtros Gammatone 18
Figura 5. Ejemplo de EMS 19
Figura 6. Diagrama de Función de Transferencia de Filtro Gammatone 21
Figura 7. MLP con una capa escondida 24
Figura 8. Tabla de intervalos de tiempo con solapamiento 26
Figura 9. Fase de entrenamiento 26
Figura 10. Señal original, Clearthroat sin ruido 28
Figura 11. Señal Clearthroat con ruido añadido EBR=6dB 28
Figura 12. Diagrama de Flujo para detección en tiempo real (realTimeDetect) 33
Figura 13. Diagrama de Flujo para detección en tiempo real (eventDetectionRealTime) 34
Figura 14. Detección en tiempo real 36
Figura 15. Gráfico de resultados por clase. NMF sin polifonía, promedio F score= 70.44 40
Figura 16. Gráfico de resultados por clase. NMF con polifonía, promedio F score= 68,23 40
Figura 17. Gráfico de resultados por clase. MLP sin polifonía, promedio F score= 75.89..... 42
Figura 18. Gráfico de resultados por clase. MLP con polifonía, promedio F score= 71.00 42
Figura 19. Tabla de resultados por clase. NMF sin polifonía con ruido añadido..... 43

LISTA DE ACRÓNIMOS

CQT	Constant Q Transform (Transformada Q Constante)
COVID-19	COronaVIrus Disease 19 (Enfermedad Coronavirus)
DFT	Discrete Fourier Transform (Transformada Discreta de Fourier)
EBR	Event to Background Ratio (Relación Evento a Base)
EMS	Envelope Modulation Spectrum (Espectro de Modulación de Envolvente)
ER	Error Rate (Tasa de Error)
IDE	Integrated Development Environment (Entorno de Desarrollo Integrado)
MFCC	Mel Frecuency Cepstral Coeficients (Coeficientes Espectrales de Frecuencias de Mel)
MLP	Multi Layer Perceptron (Discriminador Multi Capa)
NMF	Non-Negative Matrix Factorisation (Factorización Matrices No Negativas)
SVM	Support Vector Machine (Maquina de Vector Soporte)
VQT	Variable Q Transform (Transformada Q Variable)

1. INTRODUCCIÓN

El procesado de señales de audio es el tratamiento, análisis y manipulación de la información contenida en esas señales. El tratamiento de estas señales comprende diferentes técnicas: muestreo, modulación, amplificación, filtrado, codificación, etc. lo que permite el desarrollo de diferentes aplicaciones tales como la transmisión de señales en telefonía, la reducción de tamaño de almacenamiento del sonido codificado (por ejemplo música en mp3), así como aplicaciones basadas en el reconocimiento de sonidos.

Dentro de las aplicaciones de reconocimiento de sonido, destacan las aplicaciones basadas en el reconocimiento de señales de voz, que son útiles en la comunicación entre máquinas y humanos.

Ejemplo de aplicaciones basadas en reconocimiento de voz:

- Aplicaciones de red inteligente en telefonía, donde una máquina va redirigiendo al usuario final según sus respuestas telefónicas a diferentes preguntas previamente configuradas
- Asistentes personales en el hogar tales como Siri [1] (asistente virtual de Apple [2]) o Alexa [3] (asistente utilizado en la nube de Amazon [4])
- Asistentes de voz en automóviles

Pero también existen aplicaciones de reconocimiento de otro tipo de sonido, tales como aplicaciones de reconocimiento musical (Shazam [5], Soundhound [6]), o aplicaciones de identificación de escenas y eventos producidos en el entorno donde se produce el sonido (viaje en tren, canto de un pájaro, etc.).

Recientemente se han desarrollado aplicaciones capaces de realizar el diagnóstico de enfermedades, como el COVID-19, basadas en el análisis frecuencial del sonido producido por la tos del paciente y el uso de técnicas de Inteligencia Artificial. El sonido es clasificado según un modelo de redes neuronales que ha sido entrenado con los datos contenidos en una base de datos, que contiene grabaciones del sonido de la tos de miles de pacientes [7].

Las técnicas actuales para identificación de escenas de audio se basan en el estudio de la señal en el ámbito de la frecuencia. Este proyecto se ocupará de la aplicación de nuevas técnicas para la detección de escenas y eventos de sonido, determinando además el intervalo de tiempo en el que se hayan producido, basadas en el algoritmo que realiza el cálculo del EMS (Envelope Modulation Spectrum o Espectro de Modulación de Envolvente).

El EMS consigue una caracterización de la señal basada en el estudio de las modulaciones en baja frecuencia de la envolvente del sonido. Fue estudiado en trabajos científicos relacionados con la inteligibilidad de la voz para diferenciar tipos de disartria en el habla [8].

Para ello, debe realizar las siguientes funciones:

- Banco de filtros Gammatone paso banda centrados en diferentes frecuencias
- Rectificación de señal y filtro paso bajo
- Diezmado de la señal, para reducir el volumen de datos de procesado
- Cálculo del cuadrado del módulo de la DFT, para estimar el espectro de potencia
- Representación del Espectro de potencia para diferentes frecuencias de modulación de la envolvente

La realización de estas funciones está descrita en el apartado 4.1 de este informe.

La clasificación de la señal recibida es realizada mediante el uso de librerías de software que utilizan modelos de inteligencia artificial. En este proyecto se realiza la comparación entre dos modelos:

- NMF: Factorización de Matrices No Negativas [9]. Busca dos matrices no negativas (W, H) cuyo producto se aproxima a la matriz no negativa X (que caracteriza a la señal de entrada). Esta factorización se utiliza para encontrar los sonidos de los que esta señal de entrada está compuesta.
- Redes Neuronales [10]: Se basan en la utilización de un algoritmo de aprendizaje supervisado llamado "Multi Layer Perceptron (MLP)". Este algoritmo aprende una función $f(): Rm -> Ro$ a partir de un conjunto de datos de entrenamiento, donde 'm' es el número de entradas y 'o' es el número de salidas. Dado un conjunto de entradas $X = x_1, x_2, \dots, x_m$ y una salida y, puede aprender una función no lineal para clasificación de las señales. Entre la entrada y la salida puede haber una o más capas internas escondidas.

El lenguaje de programación utilizado para el desarrollo del sistema es Python 3.6. Python [11] es un lenguaje de programación interpretado, creado a finales de los ochenta, cuyos objetivos principales son la fácil lectura de su código y su ejecución en múltiples plataformas. Dispone de múltiples librerías para su utilización en aplicaciones científicas y de procesado de señal.

La evaluación del rendimiento del sistema se ha hecho según las entradas y los criterios de evaluación de la tarea nº 2 del DCASE 2016 [12], llamada *Sound event detection in synthetic audio* (detección de eventos sonoros en audio sintetizado). Los resultados están descritos en el apartado 5 de este informe, donde se puede ver que son mejores que otros sistemas realizados en MatLab.

2. MARCO TECNOLÓGICO

El sistema detector de eventos se compone de dos bloques funcionales principales: El bloque extractor de características del sonido, y el bloque clasificador.

Extracción de características

La extracción de características del evento acústico se basa habitualmente en la determinación del comportamiento de la señal en el entorno de frecuencia.

Las siguientes técnicas son de aplicación en este ámbito:

- CQT (Transformada de Q Constante) [13]: Utilizan un banco de filtros con factor de calidad (Q) igual al cociente entre la frecuencia central del filtro y su ancho de banda.
- VQT (Transformada de Q variable) [14]: Similar a CQT, pero en este caso se modifica el valor de Q según la resolución detectada por el oído humano
- MFCC (Coeficientes Cepstrales en las frecuencias de Mel) [15]: son coeficientes para la identificación del habla basados en la percepción auditiva humana. Se calculan de la siguiente forma:

1. Se separa la señal en pequeños tramos.
2. A cada tramo se le aplica la Transformada de Fourier discreta y se obtiene la potencia espectral de la señal.
3. Se aplica el banco de filtros correspondientes a la Escala Mel al espectro obtenido en el paso anterior y se suman las energías en cada uno de ellos.
4. Se toma el logaritmo de todas las energías de cada frecuencia Mel
5. Se aplica la transformada de coseno discreta a estos logaritmos.

Clasificación

La clasificación es el problema de identificar a cuál de un conjunto de tipos de señales acústicas pertenece una nueva señal, sobre la base de un conjunto de datos de señales conocidas llamada diccionario. Está considerada como un caso de aprendizaje supervisado, es decir, un aprendizaje en el que se dispone de un conjunto de observaciones correctamente identificadas.

Existen diversos algoritmos de clasificación que hacen uso de técnicas de inteligencia artificial, tales como los siguientes:

- Regresión Logística [16]: es un análisis estadístico de regresión utilizado para predecir el resultado de una muestra que puede pertenecer a diferentes clases, devolviendo la probabilidad (como valor entre 0 y 1) de que la muestra pertenezca a una clase particular.
- Máquinas de Vector Soporte (SVM) [17]: es un modelo que representa los puntos de muestra en el espacio, separando las clases en diferentes espacios. Los espacios están delimitados por el hiperplano de separación, que utiliza para su

determinación el vector soporte, cuyo objetivo es conseguir el margen máximo de separación.

- Algoritmos de Árbol de Decisión [18]: dado un conjunto de datos, se construye un diagrama de decisiones lógicas, que sirven para representar una serie de condiciones que ocurren de forma sucesiva, para la categorización final en una determinada clase.
- Bosques Aleatorios (Random Forests) [19]: es un conjunto de árboles predictores tal que cada árbol utiliza una parte independiente de los datos de muestra. La combinación de los resultados para cada árbol determinará el resultado final.
- K-Nearest Neighbours (K Vecinos más cercanos) [20]: Sirve para clasificar muestras buscando los puntos de datos más similares (y que estén más cercanos) aprendidos en la etapa de entrenamiento.

3. ESPECIFICACIONES Y RESTRICCIONES DE DISEÑO

El objetivo del proyecto es el desarrollo de un sistema que reciba como entrada una señal de audio, y pueda clasificar esta señal dentro de un conjunto de señales conocidas y un determinado intervalo de tiempo. Para ello se ha desarrollado en Python el bloque de software de extracción de características, así como dos opciones para el bloque de clasificación, basados en librerías con diferentes algoritmos: NMF y Neural Networks.

Para la evaluación de estos resultados se han utilizado los conjuntos de datos de entrenamiento, desarrollo y evaluación proporcionados por el DCASE 2016 para las fases correspondientes. El formato de salida será un fichero de texto simple (`.txt`) conteniendo los resultados del reconocimiento: instantes temporales de inicio y fin de cada evento y clase a la que pertenece el mismo. Se medirá el error cometido en la detección para la evaluación de los resultados y la comparación con el sistema base [21]. Para ello se utilizarán las medidas de ER (*Error Rate*) y F *score* [22].

De igual modo se ha desarrollado en Python un módulo para detección de sonidos en tiempo real, sobre el que se han evaluado, entre otros, parámetros de retardo en la detección. El entorno de programación seleccionado ha sido Pycharm Community 2019 3.3, y el intérprete Python 3.6.

3.1 IDE: Pycharm

El IDE (Entorno de Desarrollo Integrado) para Python elegido para el desarrollo de este proyecto ha sido Pycharm [23], desarrollado por la compañía JetBrains [24]. Se ha seleccionado Pycharm porque proporciona un amplio conjunto de facilidades para desarrollo de código, superior a otros IDE de Python existentes en el mercado. Sus características y modo de uso son descritos en este apartado.

Existen dos opciones de descarga de Pycharm, la opción Profesional y la opción Community. La opción profesional ofrece todas las características del programa, y está orientada al uso de Python para desarrollo web. Para su descarga requiere el pago de una cuota anual. La versión Community está orientada al desarrollo científico y es la versión gratuita. Se ha elegido la opción Pycharm Community 2019 3.3 para Windows, puesto que posee licencia de software libre Apache, que permite el uso del software para cualquier propósito.

La primera operación que debemos realizar en Pycharm es configurar el intérprete de Python que queremos configurar en nuestro proyecto y que habremos descargado previamente. Para desarrollar un nuevo programa en Pycharm, se debe realizar la creación de un nuevo proyecto. Allí se crearán los ficheros Python y todos aquellos que sean necesarios.

La interfaz principal de Pycharm presenta el editor de código en el centro, el explorador de archivos y directorios en la parte izquierda, y el menú en la parte superior de la pantalla, como se puede ver en la figura 1.

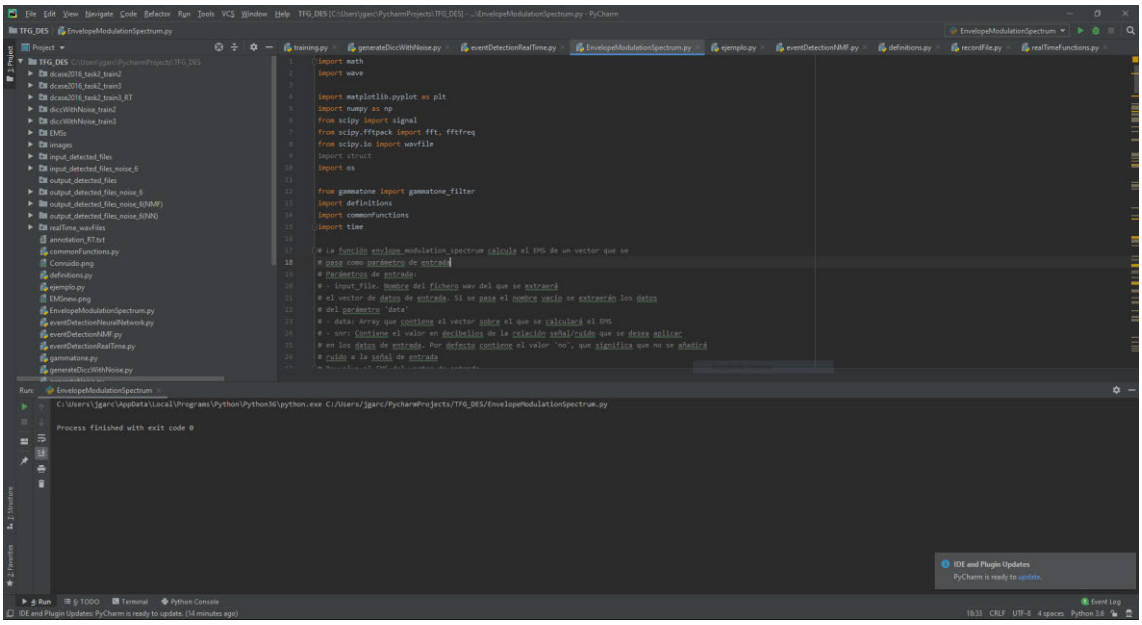

Figura 1. Interfaz principal de Pycharm

Una de las ventajas principales de Pycharm es su editor inteligente de código, que permite autocompletar los símbolos, las funciones o variables utilizadas, evitando así la generación de errores. Además posee una función de refactorización automática de código. La refactorización del código permite la re-estructuración de partes del código de modo que sea más entendible y pueda ser más eficiente sin cambiar su conducta externa. La función de refactorización puede mover partes de código, renombrar, extraer código en funciones, introducir nuevas variables o constantes, etc.

También posee un indicador de errores y warnings estáticos (análisis de código previos a la ejecución) durante la generación de código, lo que permite la edición más limpia del código fuente. Posee una función de navegación sobre el código, que nos permite saltar a los ficheros que contienen la definición de las clases, funciones o símbolos que deseemos seleccionar. Pycharm también ofrece soporte para la importación de librerías, pues puede realizar la importación e instalación de los paquetes Python necesarios asociados a un símbolo que es previamente desconocido por el intérprete de Python.

La ejecución de código se podrá realizar sencillamente dando al botón de 'run', sin necesidad de abrir ningún terminal de comandos. El progreso y el resultado de la ejecución aparecerán en una ventana de consola en la parte inferior del interfaz de Pycharm. También tiene integrado un debugger de Python y soporte para Pruebas Unitarias de código. El proyecto Pycharm puede ser integrado con sistemas de control de versión de software, tales como Subversion o git.

3.2 Lenguaje de Programación Python

El lenguaje de programación utilizado para el desarrollo de este proyecto ha sido Python, versión 3.6. Python es un lenguaje de programación creado en los años ochenta, cuya filosofía principal es la fácil legibilidad del código. Es un lenguaje interpretado, y multiplataforma. Soporta programación estructurada y orientación a objetos. Una de sus ventajas principales es la amplia disponibilidad de librerías de software libre que pueden ser importadas para su utilización. Suele ser usado en aplicaciones donde la velocidad de ejecución no es el aspecto clave.

Las principales librerías de Python utilizadas en este proyecto son las siguientes:

Scikit-learn

Scikit-learn [25] es una librería de software libre que implementa diversos algoritmos de inteligencia artificial, o análisis de datos predictivo.

Los algoritmos proporcionados están divididos en tres grupos:

1. Clasificación: se utilizan para identificar a qué categoría pertenece un objeto. Se utilizan en aplicaciones tales como detección de spam o reconocimiento de imágenes. Ejemplos de algoritmos: SVM (Máquinas de Vector Soporte), Vecinos más Cercanos, NMF (Factorización en Matrices No negativas), etc.

2. Regresión: predicen el valor de un atributo de tipo continuo, asociado a un objeto. Se utiliza en aplicaciones tales como predicción de respuesta a uso de fármacos o predicción de valores en Bolsa. Ejemplos de algoritmos: SVR (Regresión de Vector Soporte), Vecinos más Cercanos, Bosque Aleatorio, etc.

3. Agrupamiento (Clustering): realizan el agrupamiento automático de objetos similares en conjuntos. Se utiliza en aplicaciones del tipo segmentación de clientes, o agrupación de resultados de experimentos. Ejemplos de algoritmos: Medias-k (k-Means), Agrupamiento Espectral, Desplazamiento de la Media, etc.

Las librerías de clasificación utilizadas en este proyecto son:

- MLP Classifier: Clasificador basado en el algoritmo MLP (Multi Layer Perceptron o Discriminador Multi Capa, basado en redes neuronales)
- NMF: Descomposición basada en Factorización de Matrices No Negativas

Scipy

Scipy [26] es un conjunto de librerías disponibles en Python para desarrollo de aplicaciones científicas. La librería más importante de scipy utilizada en este proyecto es scipy.signal, que comprende un conjunto de funciones útiles para aplicaciones de

procesado de señal, tales como cálculo de convolución, Transformada de Fourier, diseño de filtros, análisis espectral, etc.

Numpy

Numpy [27] es una extensión de Python, que añade una biblioteca de funciones matemáticas de alto nivel para operar con vectores o matrices.

PyAudio

PyAudio [28] es una librería multiplataforma de entrada/salida de audio. Con PyAudio, podemos fácilmente utilizar Python para reproducir, grabar o procesar entradas de audio en tiempo real.

Matplotlib

Matplotlib [29] es la librería utilizada para la creación de gráficos funcionales en Python.

El listado completo de librerías de Python utilizadas y su versión correspondiente se muestra en la figura 2:

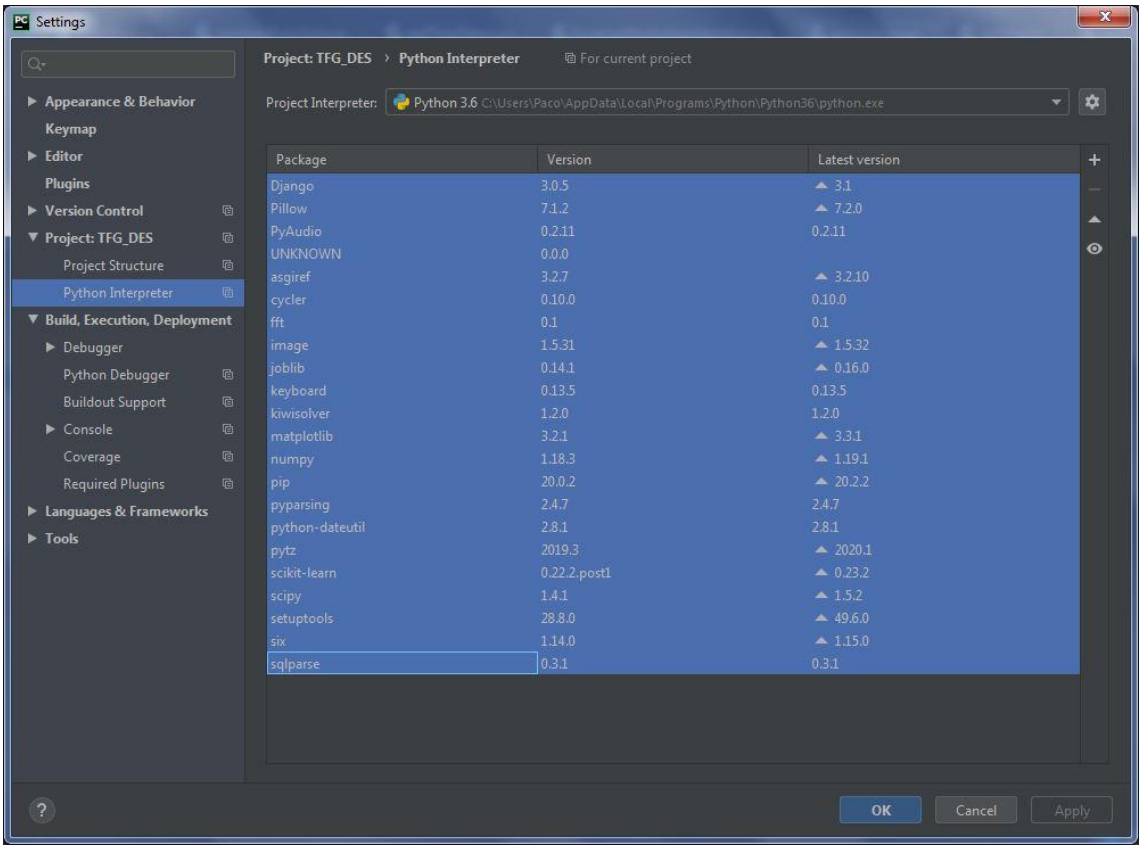

Figura 2. Listado de librerías Python utilizadas

4. DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

4.1 EXTRACCIÓN DE CARACTERÍSTICAS DE SEÑAL: EMS

Para la extracción de las características de la señal de audio que posteriormente serán clasificadas se utiliza el cálculo del Espectro de Modulación de la Envolvente (EMS).

El diagrama de bloques que representa el procesado de la señal de audio que realiza el EMS se puede ver en la figura 3.

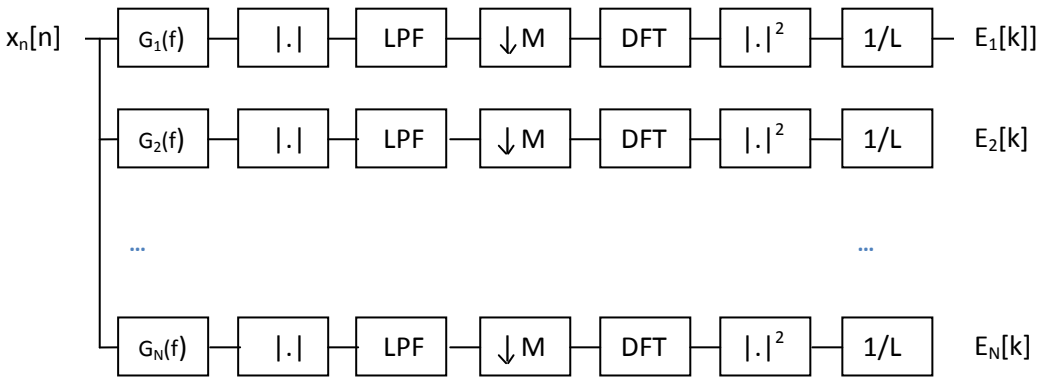

Figura 3. Diagrama de bloques del EMS

En primer lugar se realiza una normalización de la señal en potencia ($x_n\,[n]$). Una vez normalizada, se hace pasar a través de un banco de filtros Gammatone $G_N(f)$, que genera N señales, cada una de ellas conteniendo el espectro de la señal original en una banda diferente de frecuencia.

La descripción analítica de un filtro Gammatone puede verse en el siguiente apartado. La respuesta en frecuencia de un banco de filtros Gammatone se puede ver en la figura 4:

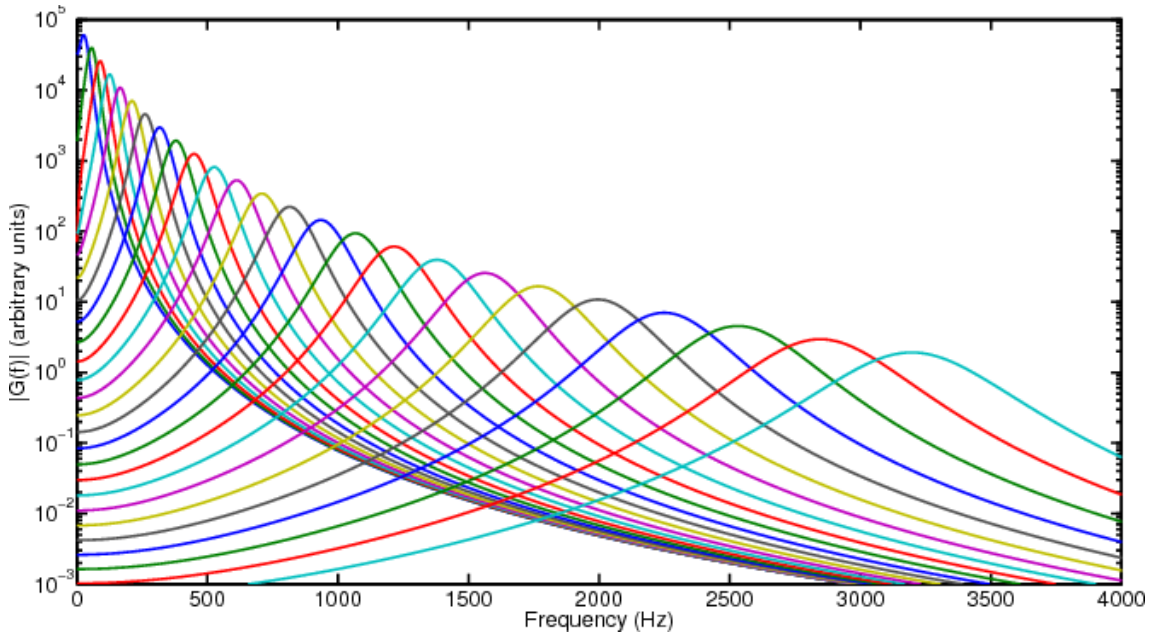

Figura 4. Respuesta en Frecuencia de Banco de Filtros Gammatone

En la presente implementación, se han utilizado 40 filtros Gammatone centrados en diferentes frecuencias que van desde 27,5 Hz hasta 17,085 kHz.

La salida de cada filtro Gammatone pasa posteriormente a través de un rectificador de señal y un filtro paso bajo. Estas operaciones nos van a permitir la obtención de la envolvente de las señales de banda de paso a la salida de los filtros. El filtro paso bajo consiste en un filtro de Butterworth con frecuencia de corte 80 Hz.

El siguiente paso es un submuestreado de la señal con un factor M=221, puesto que la señal original estaba muestreada a 44100 Hz, esto da una nueva frecuencia de muestreo de 200 Hz.

El último paso para el cómputo del EMS consiste en la estimación del espectro de potencia de la envolvente submuestreada. Para ello se calcula el módulo de la DFT y se divide por la longitud de la señal L (relación de Parseval [30]).

El EMS puede verse así como una función de dos variables: EMS(f_{mod}, f_{audio}), donde f_{mod} se refiere a la banda de frecuencia de modulación de la envolvente, y f_{audio} se refiere a cada una de las frecuencias centrales de los filtros Gammatone.

La figura 5 muestra un ejemplo de representación de EMS, con valor en escala logarítmica de colores. En el eje X se representan las bandas de modulación de frecuencia, que en este caso van en intervalos de 3 Hz, de 0 Hz a 30 Hz, y en el eje Y el número de filtros de audio Gammatone utilizados, en este caso 40.

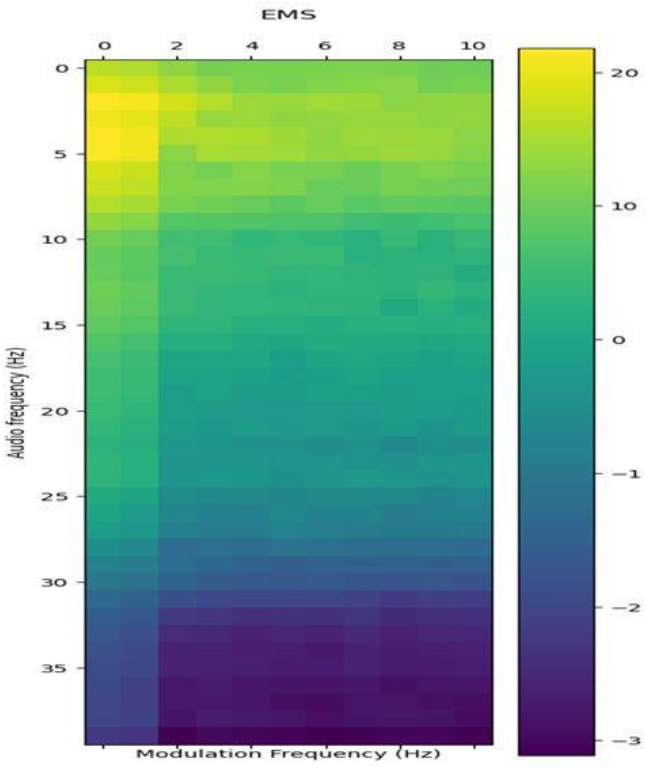

Figura 5. Ejemplo de EMS

4.2 FILTROS GAMMATONE

Un filtro Gammatone es un filtro lineal descrito por una respuesta al impulso que es el producto de una distribución gamma por un tono sinusoidal:

$$h\,(t)\;=\;at^{\,N-1}e^{-2\pi bt\cdot ERB}\cdot\;cos\,(2\pi f_c t) \tag{1}$$

A continuación se describe el cálculo analítico de la Función de Transferencia de un Filtro Gammatone [31].

Sea la función $F\,(\,\alpha\,,\,z)$ dada por la siguiente expresión:

$$F\,(\alpha,z)\;=\frac{\frac{1}{2}}{1-\,e^{\alpha}z^{-1}} \tag{2}$$

Entonces, definimos:

$$F_1\,(\alpha,z)\;=\;\frac{-z\partial F(\alpha,z)}{\partial z}\;=\;2e^{\alpha}\,F^2\,(\alpha,z)\,z^{-1} \tag{3}$$

$$F_2\,(\alpha,z) = \frac{-z\partial F_1(\alpha,z)}{\partial z} = 8e^{2\alpha}\,F^3(\alpha,z)\,z^{-2} + 2e^{\alpha}\,F^2(\alpha,z)\,z^{-1} \tag{4}$$

$$F_3\,(\alpha,z) = \frac{-z\partial F_2(\alpha,z)}{\partial z} = \\ = 48e^{3\alpha}\,F^4(\alpha,z)\,z^{-3} + 24\,e^{2\alpha}\,F^3(\alpha,z)\,z^{-2} + 2e^{\alpha}\,F^2(\alpha,z)\,z^{-1} \tag{5}$$

Si consideramos la respuesta al impulso de un filtro Gammatone:

$$h\,(t) = at^{\,N-1}e^{-2\pi bt\cdot ERB}\cdot\cos\,(2\pi f_c t) \tag{6}$$

Si $h\,(t)$ es muestreado con período de muestreo Ts:

$$h\,[n] = aT_s^{\,N-1}n^{N-1}e^{-2\pi bnTs\cdot ERB}\cdot\cos\,(2\pi f_c nT_s) = An^{N-1}e^{-\beta n}\cdot\cos\,(\Omega n) \tag{7}$$

Si $A = 1$ y $N = 4$:

$$h\,[n] = n^{3}e^{-\beta n}\cdot\cos\,(\Omega n) \tag{8}$$

Si definimos:

$$h_0[n] = e^{-\beta n}\cdot\cos\,(\Omega n) \tag{9}$$

Entonces, la transformada Z de $h\,[n]$ es:

$$H\,(z) = \frac{-z\partial}{\partial z}\left(\frac{-z\partial}{\partial z}\left(\frac{-z\partial H_0(z)}{\partial z}\right)\right) \tag{10}$$

De las tablas de pares de transformadas-z obtenemos:

$$H_0\,(z) = \frac{1 - e^{-\beta}\cos\Omega z^{-1}}{1 - 2e^{-\beta}\cos\Omega z^{-1} + e^{-2\beta}z^{-2}} \tag{11}$$

Y descomponiendo en una suma:

$$H_0\,(z) = \frac{\frac{1}{2}}{1 - e^{-\beta+j\Omega}\,z^{-1}} + \frac{\frac{1}{2}}{1 - e^{-\beta-j\Omega}\,z^{-1}} =$$

$$= F(-\beta + j\Omega, z) + F(-\beta - j\Omega, z) \tag{12}$$

Por tanto:

$$\begin{aligned} H(z) &= F_3(-\beta + j\Omega, z) + F_3(-\beta - j\Omega, z) \\ &= 48\, e^{-3\beta+j3\Omega} F^4(-\beta + j\Omega, z)\, z^{-3} + 48\, e^{-3\beta-j3\Omega} F^4(-\beta \\ &\quad - j\Omega, z)\, z^{-3} \\ &\quad + 24\, e^{-2\beta+j2\Omega} F^3(-\beta + j\Omega, z)\, z^{-2} + 24\, e^{-2\beta-j2\Omega} F^3(-\beta \\ &\quad - j\Omega, z)\, z^{-2} \\ &\quad + 2\, e^{-\beta+j\Omega} F^2(-\beta + j\Omega, z)\, z^{-1} + 2\, e^{-\beta-j\Omega} F^2(-\beta - j\Omega, z)\, z^{-1} \end{aligned} \tag{13}$$

Y podemos representar la Función de Transferencia de un Filtro Gammatone según muestra la figura 6:

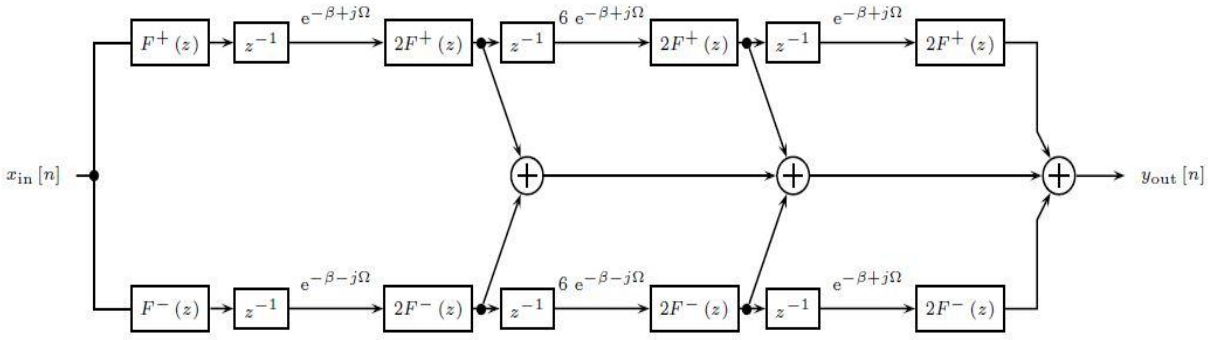

Figura 6. Diagrama de Función de Transferencia de Filtro Gammatone

4.3 MODELO DE CLASIFICACIÓN 1: NMF

A continuación se describen los principios básicos del modelo de descomposición NMF, o Factorización en Matrices No Negativas [32].

NMF se basa en la descomposición de una matriz de datos *X*, en dos matrices de rango más bajo *W* y *H*, de modo que aproximadamente se cumpla:

$$X = W\, x\, H \tag{14}$$

NMF utiliza un procedimiento iterativo para modificar los valores iniciales de *W* y *H* de modo que el producto se acerque a *X*. El procedimiento termina cuando el error de aproximación converge o se alcanza el número de iteraciones especificado.

Si suponemos que la matriz *X* se compone del conjunto de valores de una muestra, y la matriz *H* se compone del conjunto de valores de referencia (o diccionario), mediante la técnica NMF podremos obtener el conjunto de valores de la matriz *W*, que representa el peso o la componente de cada uno de los valores de referencia en nuestra muestra. Es decir, *X* representa la superposición con distintos pesos de los valores de *H*. Normalmente la matriz *H* se habrá obtenido por entrenamiento sobre un conjunto de muestras de datos de referencia.

De este modo podemos ver que el ámbito de aplicación de NMF está en la separación de fuentes o en la extracción de tópicos o componentes de una muestra, por ejemplo en la descomposición de imágenes o como en nuestro caso en la descomposición de sonidos.

El algoritmo de cálculo para NMF se centra en conseguir minimizar la distancia según la norma de Frobenius:

$$||X - WH||^2 \text{ siendo } W, H > 0 \tag{15}$$

Donde *W* y *H* son actualizadas iterativamente siguiendo esta regla:

$$H \leftarrow H \odot \frac{W^T X}{W^T W H} \tag{16}$$

$$W \leftarrow W \odot \frac{X H^T}{W H H^T} \tag{17}$$

4.4 MODELO DE CLASIFICACIÓN 2: REDES NEURONALES

Las redes neuronales artificiales se han usado ampliamente en problemas de clasificación, es decir, en problemas en los que se trata de determinar el tipo de categoría a la que pertenece un elemento desconocido.

Scikit-learn proporciona algoritmos de aprendizaje de redes neuronales tales como MLP (Multi Layer Perceptron o Discriminador Multi Capa).

MLP es un algoritmo de aprendizaje supervisado capaz de aprender una función $f(\cdot): Rm \rightarrow Ro$ por entrenamiento sobre un conjunto de datos, donde 'm' es el número de dimensiones para la entrada y 'o' es el número de dimensiones para la salida.

Dado un conjunto de funciones o características $X = x_1, x_2, \ldots, x_m$ y una salida 'y', MLP puede aprender una aproximación a una función no lineal para clasificación o regresión. Entre la entrada y la salida puede haber una o más capas llamadas capas escondidas.

La figura 7 muestra un MLP con una capa escondida.

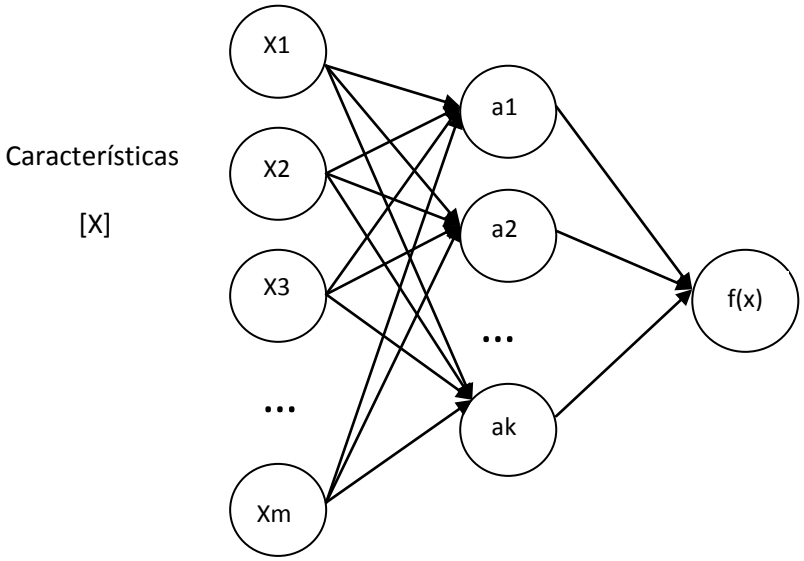

Figura 7. MLP con una capa escondida

La capa que está a la izquierda, conocida como la capa de entrada, consiste de un conjunto de neuronas $\{x_i|x_1, x_2, \ldots, x_m\}$ que representan las características de entrada. Cada neurona de la capa escondida transforma los valores de la capa previa con una suma ponderada lineal $w_1x_1 + w_2x_2 + \ldots + w_mx_m$, seguida de una función de activación no lineal $g(\cdot): R \rightarrow R$ (como la función tangente hiperbólica). La capa de salida recibe los valores de la última capa escondida y los transforma en valores de salida.

Las ventajas de MLP son la capacidad de aprender modelos no lineales y la capacidad de aprender modelos en tiempo real. Sus desventajas son que la exactitud de validación depende de inicializaciones aleatorias y puede variar, que requiere el ajuste de parámetros tales como número de capas intermedias o número de iteraciones y que es sensible a escalado de las características de entrada.

4.5 ARQUITECTURA DEL SISTEMA

El sistema se compone de dos fases principales, la fase de Entrenamiento y la fase de Detección.

Se ha dividido la fase de entrenamiento en tres secciones:

- Entrenamiento: Fase general que sirve tanto para NMF como para Neural Network
- Entrenamiento con ruido añadido: Se añade ruido a los ficheros originales

- Entrenamiento para la detección en tiempo real: Donde se genera un nuevo diccionario con nuevas grabaciones

A su vez, se ha dividido la fase de Detección en tres secciones diferentes:

- Detección NMF: sobre ficheros grabados, basada en el algoritmo de clasificación NMF
- Detección Neural Network: sobre ficheros grabados, que utiliza un algoritmo de Redes Neuronales en la detección
- Detección en tiempo real: sobre entrada de audio capturada por el micrófono en tiempo real, usando como algoritmo de clasificación el que obtuvo mejor resultado entre los anteriores

4.5.1 Entrenamiento

La fase de entrenamiento tiene como objetivo generar un conjunto de datos de referencia que nos puedan servir para realizar la clasificación posterior.

En el caso del "challenge" de la task2 de DCASE 2016, se proporcionan un conjunto básico de ficheros de sonido de "training" que podrán ser utilizados para la detección de otros sonidos similares.

En total se proporcionan 221 archivos de longitud variable (entre 0.5 y 3 segundos), pertenecientes a 11 clases diferentes: "clearthroat, cough, doorslam, drawer, keys, keyboard, knock, laughter, pageturn, phone, speech".

En base a la longitud media de estos ficheros se decide realizar un cálculo homogéneo para el EMS de ficheros de 1 segundo de duración. Esto requerirá cortar los ficheros de entrenamiento en pedazos de 1 segundo de duración, en caso de que su duración total sea mayor de 1 segundo. Esto se consigue con el módulo Python "split.py", que toma como entrada todos los ficheros contenidos en el path definido en el parámetro "inputPath", contenido en el fichero Python "definitions.py".

Se observa, además, que en el caso de los ficheros de desarrollo, los sonidos a detectar están separados por tramos de ruido Gaussiano, que no aparece en los ficheros de entrenamiento. Para evitar que el sistema detector confunda estos tramos de ruido con las clases de sonidos a detectar, se decide generar ficheros de ruido Gaussiano con una señal aleatoria, de 1 segundo de duración, e integrarlos con el resto de clases de sonido. De modo que se genera una nueva clase auxiliar que hemos llamado "noise".

En la partición de ficheros a ficheros de máximo 1 segundo de duración, se produce un solapamiento de 500 ms, de modo que por ejemplo un fichero original de 2.3 segundos producirá tras su partición 4 ficheros que comprenderán los siguientes intervalos:

Figura 8. Tabla de intervalos de tiempo con solapamiento

Fichero	Intervalo
1	0.0s – 1.0s
2	0.5s – 1.5s
3	1.0s – 2.0s
4	1.5s – 2.3s

Una vez obtenido el conjunto de ficheros de 1 segundo que será utilizado como referencia (si es necesario se rellenan con ceros), se procede al cálculo del EMS de todos ellos (554 archivos en total), mediante el uso del módulo Python "training.py".

Si utilizamos como referencia 10 bandas de frecuencia entre 0 y 30 Hz, la longitud del EMS será de 40 frecuencias Gammatone X (10+1) bandas = 440. Como tenemos 554 archivos, habremos generado una matriz de referencia de dimensiones (554 x 440) que llamaremos *H*, o matriz diccionario.

La fase de entrenamiento se puede observar en la figura 8:

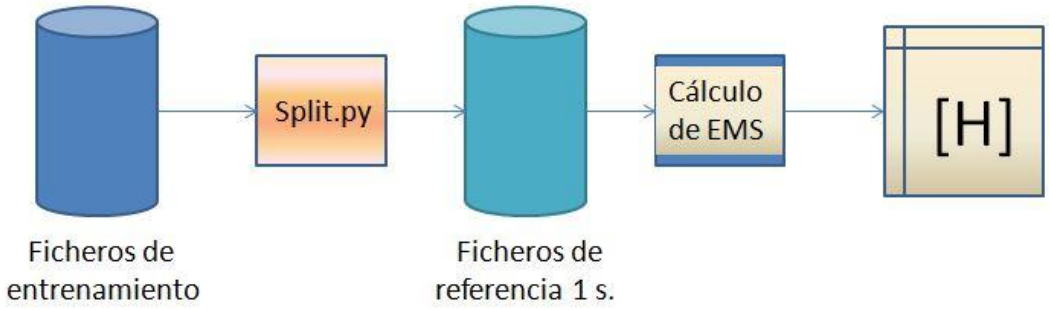

Figura 9. Fase de entrenamiento

4.5.2 Entrenamiento con ruido añadido

Una característica de los ficheros proporcionados para el DCASE challenge, es que los ficheros de entrenamiento están grabados en diferentes condiciones respecto a los archivos de desarrollo. Los ficheros de entrenamiento son ficheros grabados sin ruido, mientras que los ficheros de desarrollo son ficheros grabados con tres niveles de ruido: relación señal ruido de 6 decibelios, 0 decibelios y -6 decibelios. Para intentar conseguir una mayor igualdad entre los ficheros de entrenamiento y los de desarrollo, se realizará también la suma artificial de una señal de ruido a los ficheros de entrenamiento.

A continuación se describe la forma de añadir diferentes niveles de ruido gaussiano a estos archivos.

Supongamos que deseamos añadir una señal de ruido $n[n]$ a una señal de referencia $s[n]$, de modo que la relación señal ruido (S/N) en decibelios sea *EBR* dB. En nuestro caso, *EBR* podrá tomar los valores 6, 0 ó -6.

Dado que la señal $s[n]$, es conocida (proporcionada en el fichero de sonido para el entrenamiento), podemos calcular su potencia media en Watios como $Sw = \frac{\sum s^2[n]}{n}$, y en decibelios como $S = 10\ log\ \frac{\sum s^2[n]}{n}$.

La potencia media de ruido se puede calcular como $N = S - EBR$ decibelios.

Como $N = 10\ log\ Nw$, la potencia media de la señal de ruido en watios será: $Nw = 10^{\frac{N}{10}}$

Para generar la señal de ruido n, podemos utilizar una función de Python para generación de señales aleatorias de media = 0, desviación típica $\sigma = \sqrt{N_w}$ y n muestras.

La función utilizada es la siguiente:

numpy.random.normal(*loc=0.0*, *scale=1.0*, *size=None*)

que genera muestras aleatorias de una distribución Normal o Gaussiana

Los parámetros de entrada son:

loc: media de la distribución (cero en el caso de ruido gaussiano)

scale: desviación estándar de la distribución ($\sqrt{N_w}$ para la generación de ruido)

size: tamaño de la señal (número de muestras *n* de la señal $s[n]$)

La señal de referencia con ruido s_n se formará entonces haciendo la suma de la señal original con esta señal aleatoria de ruido *n*:

$$s_n[n] = s[n] + n\,[n]$$

En el ejemplo (ver figuras 9 y 10) se puede apreciar una señal de referencia original, y una señal de referencia con ruido añadido de forma que la relación señal ruido es *EBR=6dB*

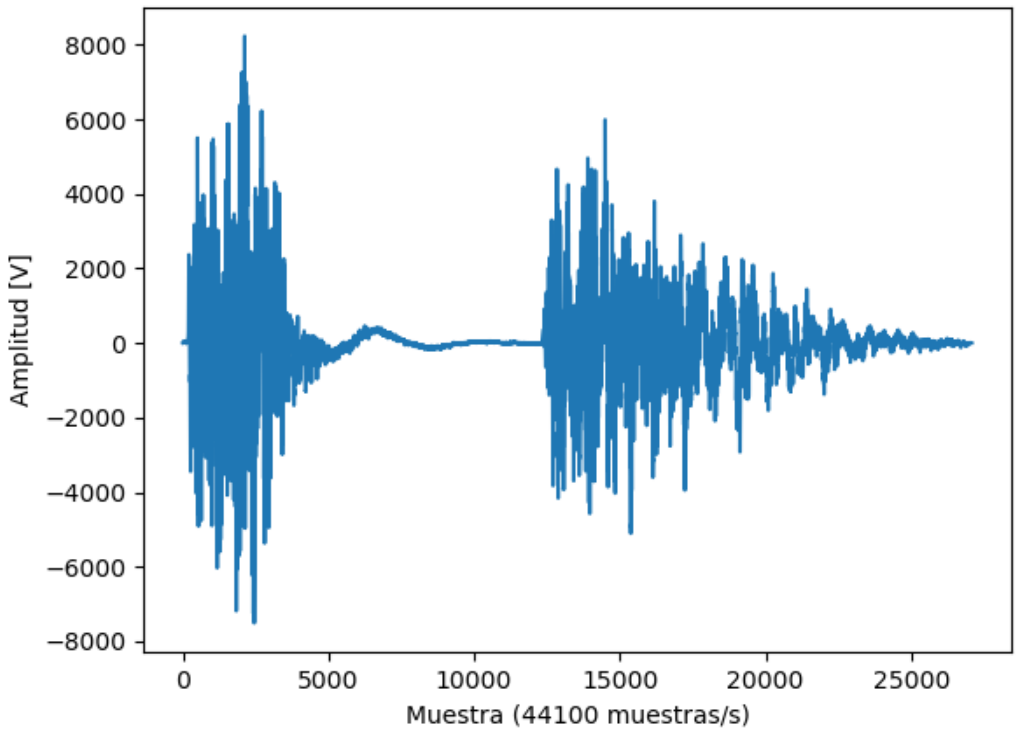

Figura 10. Señal original, Clearthroat sin ruido

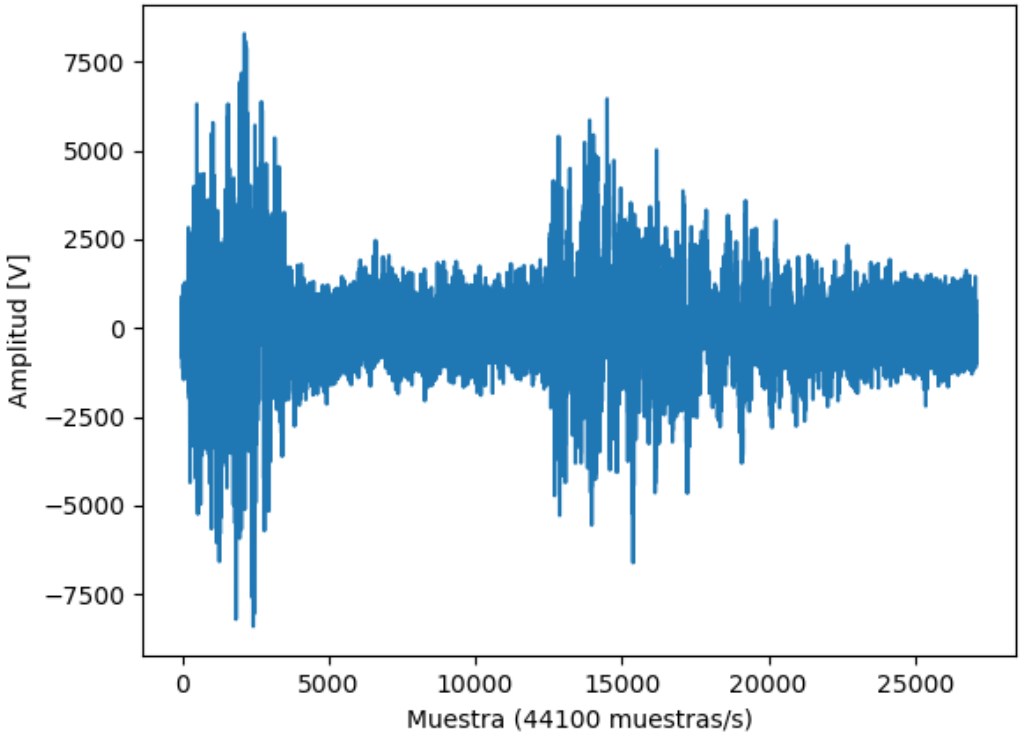

Figura 11. Señal Clearthroat con ruido añadido EBR=6dB

La estrategia utilizada ha sido incluir en el conjunto se señales de referencia, tanto las señales originales como las señales obtenidas mediante adición de ruido. El resultado de la evaluación para señales de referencia con ruido añadido puede verse en el capítulo 5.

4.5.3 Entrenamiento para la detección en tiempo real

Para el caso de la detección en tiempo real, no disponemos de un conjunto de ficheros de entrenamiento, de modo que se procederá a realizar grabaciones de sonidos similares a los que posteriormente se detectarán en tiempo real. Para ello se ha diseñado el módulo Python "recordFile.py", que graba el sonido con los mismos parámetros con los que será obtenido a través del micrófono.

Al ejecutar este módulo, nos preguntará por el nombre del fichero a grabar, que deberá contener el nombre de alguna de las clase de sonido a detectar (definidas en "classes_RT"), y por el número de segundos que durará la grabación. En este caso las grabaciones se realizarán en ausencia de ruido, por lo que no será necesario el uso de una clase auxiliar "noise".

Este módulo Python hace uso de la librería PyAudio para capturar la entrada del micrófono del ordenador, disponible en Python. Con "PyAudio", en primer lugar se abre un "stream" de datos de entrada con los parámetros deseados (44100 muestras por segundo, 1 canal de audio, etc). Posteriormente se irán leyendo las tramas de entrada en grupos de 1024, para finalmente, una vez que ha transcurrido el número de segundos establecidos almacenar la información leída en un fichero formato *wav* con el nombre obtenido.

Una vez que se dispone de las grabaciones necesarias, se hará uso del módulo Python "split.py", que realizará una partición de los ficheros grabados en ficheros de duración 1 segundo, haciendo además un solapamiento de 500 ms entre ellos. De este modo, una grabación de un sonido de, por ejemplo, 2.3 segundos, nos permitirá la generación de 4 ficheros solapados de 1 segundo de duración, que compondrán 4 entradas en el diccionario.

Finalmente, el módulo "training.py" nos permitirá generar la matriz diccionario H para detección en tiempo real. Este módulo tomará como entrada cada uno de los ficheros de un segundo de duración generados por el módulo "Split.py", calculará el EMS de la señal contenida en estos ficheros, y generará una matriz con el resultado.

4.5.4. Detección NMF

El módulo Python "eventDetectionNMF.py" realiza la detección en base al algoritmo NMF (Non-Negative Matrix Factorisation). Este módulo utiliza la librería Python de scikit-learn para crear un modelo NMF.

En primer lugar se carga la matriz diccionario *H*, y se escala mediante el uso de un standardScaler de la librería de scikit-learn.

El resultado del escalado 'z' para un valor 'x' es el siguiente:

$$z = (x - u) / s$$

donde 'u' es la media de todos los valores o cero si se setea el valor 'with_mean=False', y 's' es la desviación estándar.

En el caso de escalado para NMF, no se resta la media a cada valor puesto que hay que evitar valores negativos.

Según la nomenclatura NMF el número de componentes del modelo será el número total de ficheros del diccionario, y el número de "features" será cada uno de los valores del EMS.

El modo de aprender el modelo será cargando en él la matriz diccionario H que habíamos calculado anteriormente en la fase de entrenamiento (método model.fit()).

El conjunto de entrenamiento proporcionado en DCASE consiste de 18 ficheros *wav* de 120 segundos de duración, grabados en distintas condiciones de ruido y solapamiento. El nombre de los ficheros tiene el formato *'dev_1_ebr_N_nec_X_poly_P.wav'*, siendo *N* el EBR de la señal, *X* el número de eventos por cada clase y *P* la presencia (1) o ausencia (0) de polifonía.

Posteriormente van leyendo en secuencia cada uno de los ficheros de este conjunto. Este se divide en intervalos de 1 segundo. Para mejorar la detección, esta división se hace en intervalos solapados 500 ms entre sí.

Se calcula el EMS de cada intervalo, EMS=Xi

Se introduce en el modelo (método model.transform()), de modo que éste nos devuelve la matriz de componentes *Wi* que cumple el producto de matrices:

$$Xi = Wi * H$$

Wi contiene entonces los coeficientes que representan componentes de cada clase de sonido, de modo que si se trata por ejemplo de "clearthroat", el primer componente de *Wi* será próximo a '1' y el resto serán idealmente próximos a '0'.

En realidad, como disponemos de varios ficheros de cada clase (por ejemplo 20 ficheros de clearthroat), la suma de esos primeros 20 coeficientes es el valor que será próximo a '1'.

Cuando para un determinado intervalo se obtiene un valor de componente de una clase próximo a ´1´o superior a un umbral definido, se anota en un fichero el tiempo de comienzo de este evento, de igual modo que se anota su finalización en el inicio del intervalo cuando deja de aparecer.

También se evalúa la componente de la clase auxiliar "noise", de modo que cuando esta componente es superior a un umbral se decide que se trata de un intervalo de sólo ruido y no se realiza la evaluación de las componentes del resto de clases.

En la detección se han observado dos clases con características especiales: "keys y pageturn". En el caso de "keys", se acepta la componente de esta clase cuando es superior al umbral incluso en el caso de que la componente de ruido sea alta, pues se ha observado este efecto en la detección, debido a que la duración del sonido es muy inferior a 1 segundo. En el caso de "pageturn", esta clase se confunde con ruido (el EMS de "pageturn" es similar al EMS de "noise") y se ponen requerimientos más estrictos en cuanto a que la componente de ruido debe ser más baja que en las demás clases para proceder a la evaluación de su componente. Es decir, si la componente de pageturn es alta, pero la componente de ruido no es muy baja, se desecha en este intervalo.

Este fichero de texto con el resultado sigue el formato definido por el DCASE 2016 task 2, y es el que será utilizado para evaluar el resultado de la detección, mediante un módulo de MatLab ya preparado por la organización de DCASE.

Los resultados se analizan en el apartado 5.

4.5.5. Detección Neural Network

El módulo Python "eventDetectionNN.py" realiza la detección en base al algoritmo MLP (Multi Layer Perceptron) que utiliza redes neuronales para su implementación.

Este módulo utiliza la librería Python de scikit-learn para crear un modelo MLPClassifier. El modelo se crea con una capa intermedia escondida de 100 neuronas.

La red neuronal estará formada además por 440 neuronas a la entrada (una por cada dígito del EMS: 10 +1 bandas x 40 frecuencias Gammatone) y 12 neuronas a la salida (una por cada clase de sonido más la clase auxiliar de ruido).

En primer lugar se carga la matriz diccionario H, y se normaliza mediante el uso de un standardScaler de la librería de scikit-learn. Posteriormente se introducen los datos de la matriz diccionario en el modelo, haciendo corresponder a cada entrada del diccionario

con su salida esperada en el array output_neurons-dictionary. Por ejemplo, al EMS de un fichero clearthroat se le hará corresponder el array de salida [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

También se hace uso en este modelo de la clase auxiliar "noise", con sus ficheros *wav* correspondientes generados con señal aleatoria. De igual modo, se tratan de la misma forma que en el caso NMF las clases "keys" y "pageturn".

Posteriormente van leyendo en secuencia cada uno de los ficheros de detección del conjunto de entrenamiento de DCASE. Este se divide en intervalos de 1 segundo. Para mejorar la detección, esta división se hace en intervalos de tiempo solapados 500 ms entre sí.

Se calcula el EMS de cada intervalo, y se obtiene el array de neuronas de salida que da el modelo MLP para esta entrada. Si el valor de una neurona correspondiente a una determinada clase de sonido es superior a un umbral previamente definido, se anota evento de comienzo para esa clase. De igual modo, cuando el valor de la neurona sea inferior al umbral, se anotará la finalización del evento.

Estas anotaciones se realizan en un fichero de texto con el resultado que sigue el formato definido por el DCASE 2016 task 2.

Los resultados se analizan en el apartado 5, donde se puede comprobar que el modelo MLP da en este caso mejores resultados que el modelo NMF.

4.5.6. Detección Tiempo Real

La detección en tiempo real (ver figura 11) utiliza un modelo MLPClassifier para reconocimiento de sonidos. En primer lugar, se carga en el modelo el diccionario que hemos obtenido previamente con la grabación de ficheros de audio y la evaluación de sus correspondientes EMS.

En el caso de la detección en tiempo real, en vez de tomar la entrada de sonido a detectar de un conjunto de ficheros *wav*, se tomará directamente de la entrada de micrófono del ordenador. Esta entrada se tomará en intervalos de 1 segundo.

Además de realizar la detección de sonido, este módulo muestra cada segundo de la señal de entrada en un plot en tiempo real, y genera como salida un icono con la imagen del sonido detectado y un fichero de texto con los intervalos de tiempo y el nombre de las clases de sonido detectado en el formato definido por el DCASE challenge.

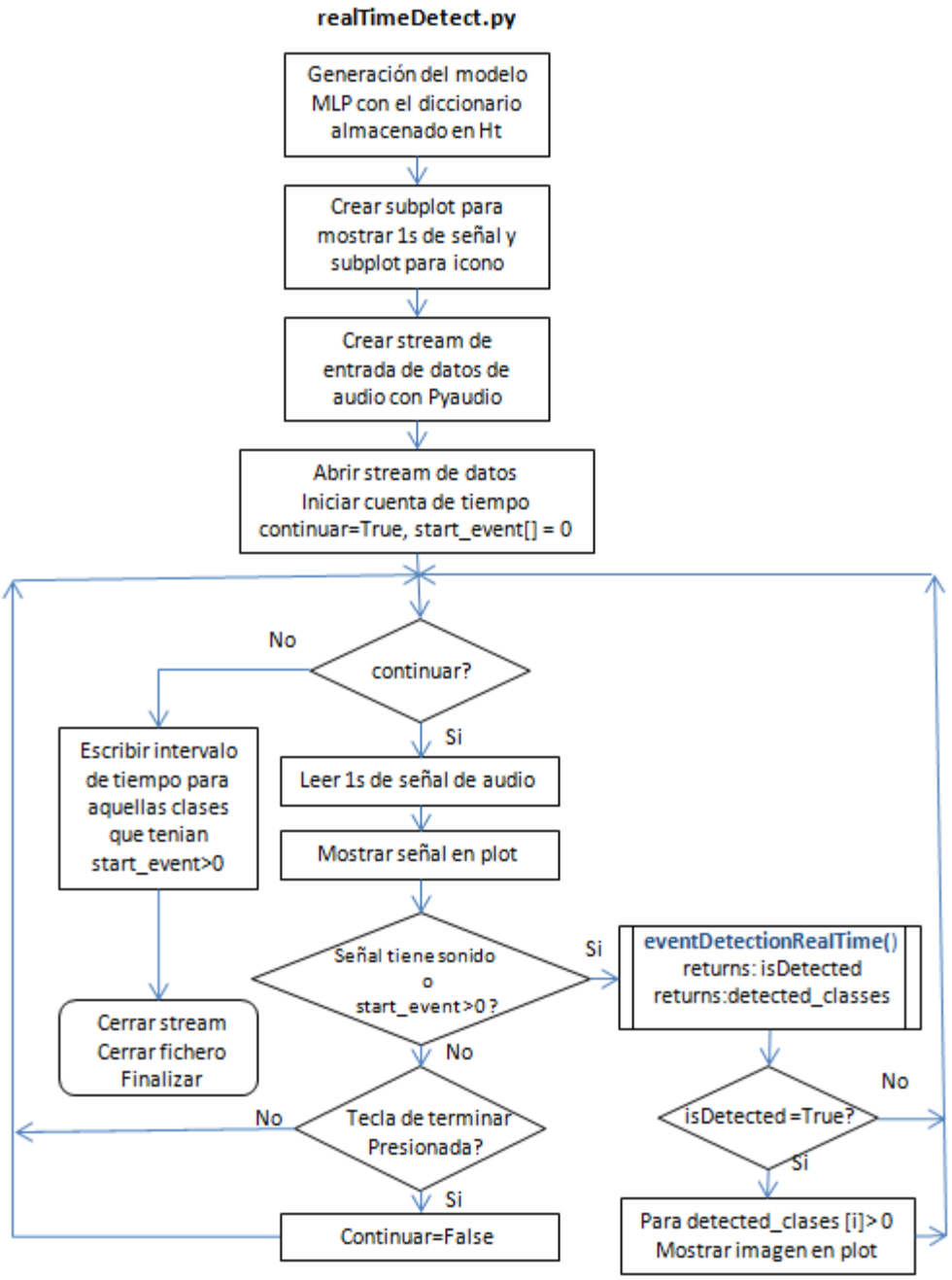

Figura 12. Diagrama de Flujo para detección en tiempo real (realTimeDetect)

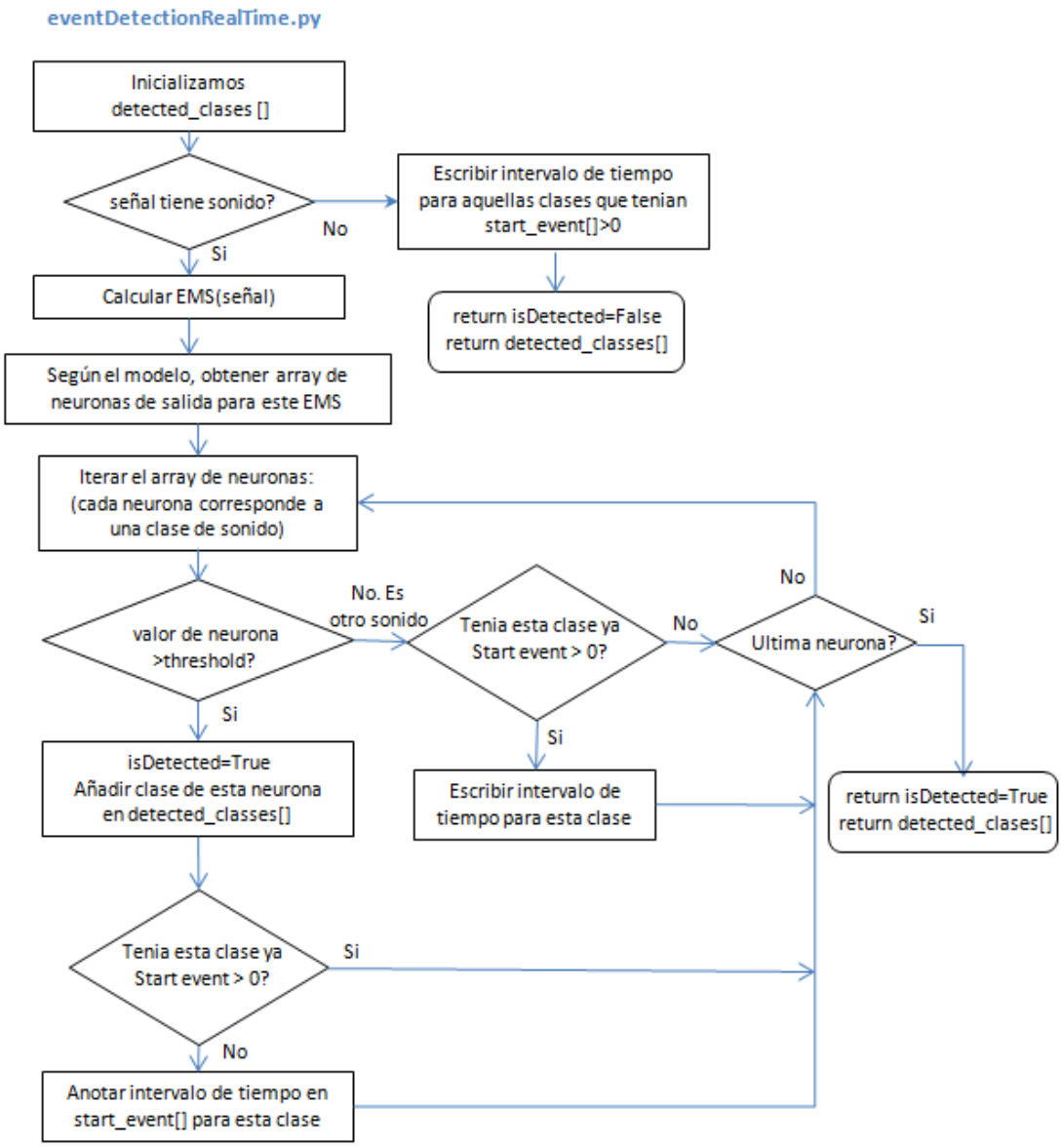

Figura 13. Diagrama de Flujo para detección en tiempo real (eventDetectionRealTime)

La implementación de la detección en tiempo real en Python se compone de dos módulos principales: realTimeDetect.py y eventDetectionRealTime.py.

realTimeDetect.py es el módulo que será invocado por el usuario para iniciar la detección en tiempo real. Este módulo generará en primer lugar un modelo MLP tomando como entrada el fichero con el diccionario que habremos generado en la fase de entrenamiento. Posteriormente creará dos subplots, uno para mostrar un segundo de señal, y otro para mostrar los iconos que corresponden a los sonidos detectados. Abrirá un stream de datos de entrada procedentes de la entrada de audio, inicializará un array llamado "start_event" con ceros e iniciará la cuenta de tiempo. El array "start_event" contendrá, para cada una de las clases de sonido, los intervalos de tiempo en los que se

ha detectado el comienzo de un sonido de esa clase. A continuación se iniciará el bucle principal del módulo, que no finalizará hasta que el usuario presione la tecla de finalización del programa. En este bucle se mostrará un segundo de señal de entrada en el plot correspondiente. Si este segundo de señal tiene sonido (se considera así si un número predeterminado de muestras tiene un valor superior a un umbral mínimo, parámetros definidos en el módulo definitions.py), o si ya habíamos detectado algún evento de sonido previamente, se llamará a la función "eventDetectionRealTime()", contenida en el módulo Python "eventDetectionRealTime.py". Esta función nos devolverá dos parámetros: "is_detected", que tendrá valor True si se ha detectado algún sonido, y el array "detected_clases" que contendrá la lista de clases de sonido con evento activo durante este intervalo. Si el valor de "is_detected" es *True*, se procederá a mostrar los iconos de las clases detectadas en el plot de imágenes.

"eventDetectionRealTime.py" es el módulo Python que contiene la función invocada por "realTimeDetect.py" para devolver los parámetros "is_detected" y "detected_clases". En primer lugar comprobará si este segundo de señal tiene sonido, pues si no es así simplemente escribirá en el fichero de resultado los instantes inicial y final de aquellas clases que tenían evento detectado, y reinicializará el array "detected_clases" a cero. Devolverá el parámetro is_detected con el valor "False". En el caso de que el segundo de señal procesado contenga sonido, procederá al cálculo de su EMS. Este EMS será introducido como entrada en el modelo MLP, para obtener el array de neuronas de salida, que contendrá la probabilidad de presencia de cada clase de sonido correspondiente a este EMS. Se iterará a través de este array, para comprobar si cada probabilidad es superior al umbral definido para la detección, y en ese caso se pondrá la variable "is_detected" a True y se anotará el intervalo de tiempo en el array "detected_clases", en el índice correspondiente a la clase detectada. Finalmente retornará el valor de estos parámetros.

El proceso dura en un ordenador de sobremesa unos 4 segundos, que es el tiempo que tarda en aparecer el icono una vez que se ha producido el sonido.

El programa presenta en pantalla la reproducción en tiempo real de la señal recibida por la entrada de audio y un icono que representa el último sonido detectado, como se puede apreciar en la figura 14.

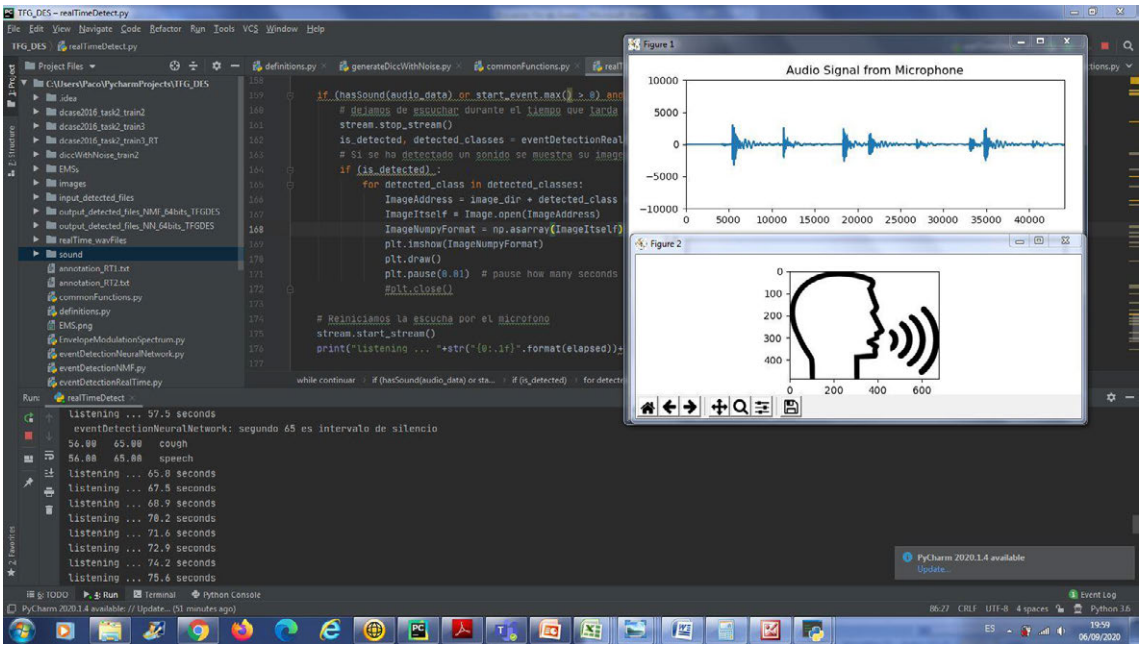

Figura 14. Detección en tiempo real

5. RESULTADOS

5.1 EVALUACION DE RESULTADOS

Las medidas utilizadas para la evaluación de los resultados son las mismas que las que han sido propuestas en el challenge DCASE 2016. La descripción detallada y el procedimiento de cálculo de las métricas se presenta en [33].

Para hacer los cálculos se realiza una comparación entre los ficheros de texto generados por los programas de detección en Python, con los resultados previstos en los ficheros de anotación.

La métrica principal utilizada para la evaluación es el ratio de error basado en segmento.

Las métricas son Precision (P), Recall o exhaustividad (R) y F-Score (F), que es la medida global del error.

Se definen como:

$$P = \frac{TP}{TP + FP}$$

TP: número de positivos verdaderos

FP: número de falsos positivos

$$R = \frac{TP}{TP + FN}$$

TP: número de positivos verdaderos

FN: número de falsos negativos

$F = \frac{2\,P\,R}{P+R}$ o, alternativamente, basándose en las medidas intermedias:

$$F = \frac{2\,TP}{2\,TP + FP + FN}$$

También se mide la tasa de error (ER) como el número de errores en detección en base a las sustituciones, eliminaciones e inserciones.

Sustituciones (S): es el número de eventos que han sido detectados pero que no se corresponden con la clase correspondiente:

$$S(k) = mín\,(FN(k), FP(k))$$

FP: número de falsos positivos

FN: número de falsos negativos

k: número de segmento

Deletions, o Borrados (D): es el número de eventos que no han sido detectados:

$$D(k) = máx\,(0, FN(k) - FP(k))$$

FP: número de falsos positivos

FN: número de falsos negativos

k: número de segmento

Inserciones (I): es el número de eventos que han sido detectados pero que no son correctos:

$$I(k) = máx\,(0, FP(k) - FN(k))$$

FP: número de falsos positivos

FN: número de falsos negativos

k: número de segmento

El Ratio de Error (ER) se define finalmente como:

$$ER = \frac{\sum_{k=1}^{k} S(k) + \sum_{k=1}^{k} D(k) + \sum_{k=1}^{k} I(k)}{\sum_{k=1}^{k} N(k)}$$

Siendo *N* el número de eventos activos en el segmento *k*.

Estas métricas pueden ser obtenidas de dos formas: en base a segmento y en base a eventos.

En la primera de ellas, la obtención de las medidas se realiza segmento por segmento, en segmentos temporales de duración fija.

En la segunda, los resultados se comparan evento a evento, siendo válidos cuando sus posiciones temporales se superponen con una tolerancia de 200 ms para el tiempo de inicio y 200 ms o la mitad de la longitud del evento para el tiempo de fin. Para obtener buen resultado en la detección basada en eventos es necesario tener una mayor precisión temporal.

El conjunto de resultados es obtenido además para cada una de las clases y de forma global, con la media de todas ellas.

En el análisis de los resultados del DCASE 2016 [34] se tiene en cuenta el valor-F en base a eventos como principal medida del rendimiento del sistema, y el ER como medida secundaria.

La evaluación se realiza a través de la siguiente función de MatLab:

Evaluate. Evalúa los resultados obtenidos en la detección comparando los ficheros *.txt* de salida y las anotaciones en base a las métricas descritas anteriormente, haciendo uso de la librería *'metrics'* incluida en el código del sistema base del DCASE 2016.

Parámetros de entrada:
sResultsFolder: ruta relativa a la carpeta que contiene los ficheros *.txt* de salida de la detección.
sReferenceFolder: ruta relativa a la carpeta que contiene los ficheros *.txt* de anotaciones manuales.

Parámetros de salida:
eSegmentBased: datos de los resultados basados en segmentos.
eEventBased: datos de los resultados basados en eventos.

5.2 PRUEBAS REALIZADAS

A continuación de presentan los resultados de las distintas pruebas realizadas. En todas ellas el algoritmo detector utilizado ha sido el EMS.

Como referencia de comparación, el sistema base proporcionado como ejemplo en DCASE obtuvo un resultado de F-score = 41.6% y ER = 0.7859 para detección basada en segmento. Otros sistemas desarrollados en MatLab con algoritmo clasificador NMF han obtenido F-score = 69.1% ER = 0.5937 (sin polifonía).

5.2.1 Algoritmo de clasificación NMF

En este caso, el algoritmo de clasificación utilizado ha sido NMF.

Los ficheros utilizados para la detección y las reglas de evaluación son los proporcionados por DCASE Challenge 2016.

Las condiciones utilizadas para el cálculo de EMS han sido:

- Bandas de sonido: 0 a 30 Hz, en intervalos de 3Hz
- Número de Filtros Gammatone: 40

Se ha usado una clase auxiliar de ruido para eliminar intervalos de “solo ruido”.

Dividimos los resultados en dos grupos: ficheros sólo sin polifonía y todos los ficheros (incluyendo aquellos que contienen polifonía).

Resultado sin polifonía:

- Segmento: F-Score= 70.44%, ER: 0.5397

Los resultados por clase de sonido (Segmento, F score - ER) se muestran en la figura 15.

Resultado con Polifonía:

- Segmento: F-Score= 68,23%, ER: 0.5522

Los resultados por clase de sonido (Segmento, F Score- ER) se muestran en la figura 16.

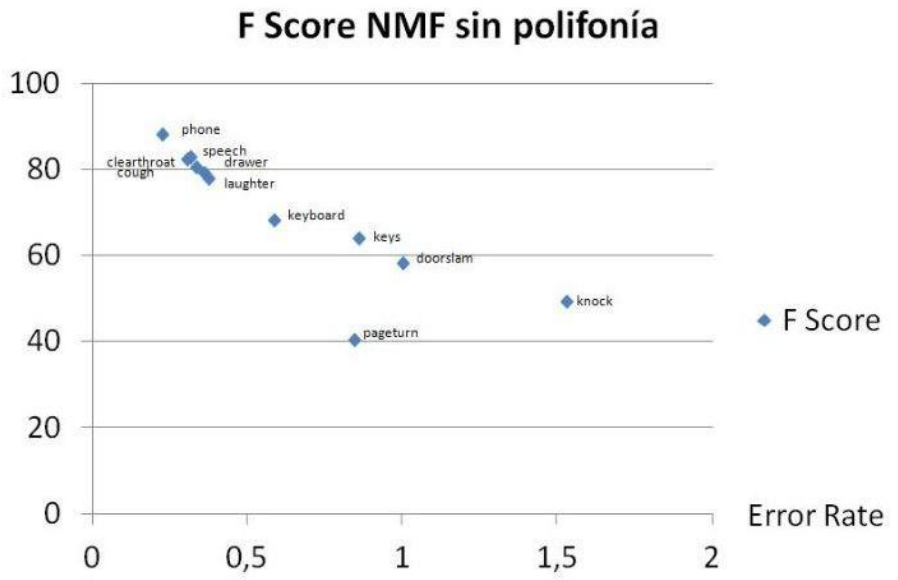

Figura 15. Gráfico de resultados por clase. NMF sin polifonía, promedio F score= 70.44

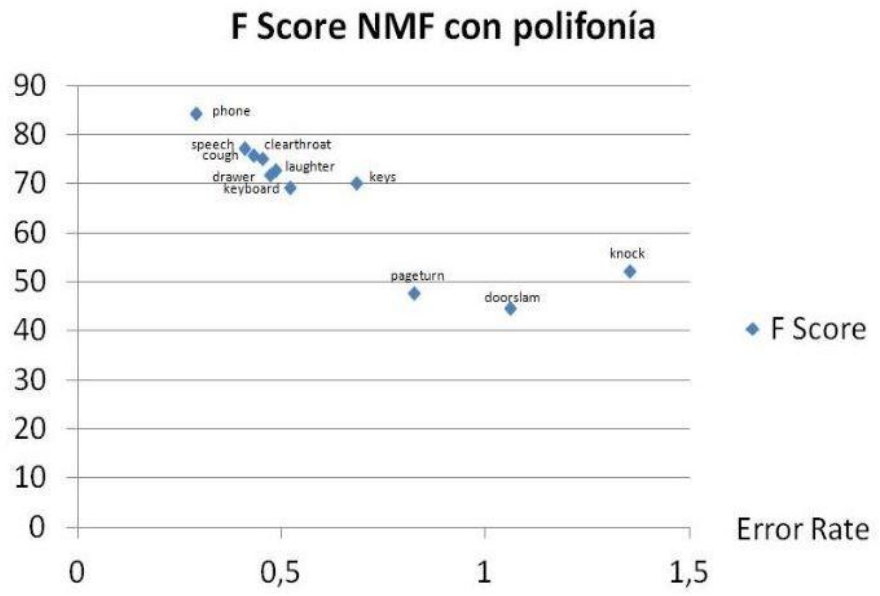

Figura 16. Gráfico de resultados por clase. NMF con polifonía, promedio F score= 68,23

5.2.2 Algoritmo de clasificación MLP (Neural Networks)

En este caso, el algoritmo de clasificación utilizado ha sido MLP.

Los ficheros utilizados para la detección y las reglas de evaluación son los proporcionados por DCASE Challenge 2016.

Las condiciones utilizadas para el cálculo de EMS han sido:

- Bandas de sonido: 0 a 30 Hz, en intervalos de 3Hz
- Número de Filtros Gammatone: 40

Se ha usado una clase auxiliar de ruido para eliminar intervalos de "solo ruido".

Se han introducido una capa intermedia de 100 neuronas en la red neuronal.

Dividimos los resultados en dos grupos: ficheros sólo sin polifonía y todos los ficheros (incluyendo aquellos que contienen polifonía).

Resultado sin polifonía:

- Segmento: F-Score= 75.89%, ER: 0.4263

Los resultados por clase de sonido (Segmento, F score - ER) se muestran en la figura 17.

Resultado con Polifonía:

- Segmento: F-Score= 71.00%, ER: 0.4871

Los resultados por clase de sonido (Segmento, F score - ER) se muestran en la figura 18.

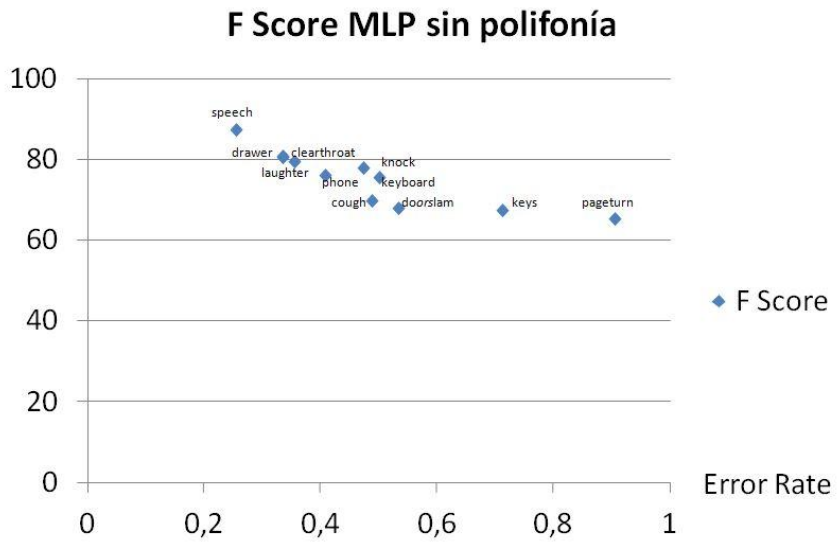

Figura 17. Gráfico de resultados por clase. MLP sin polifonía, promedio F score= 75.89

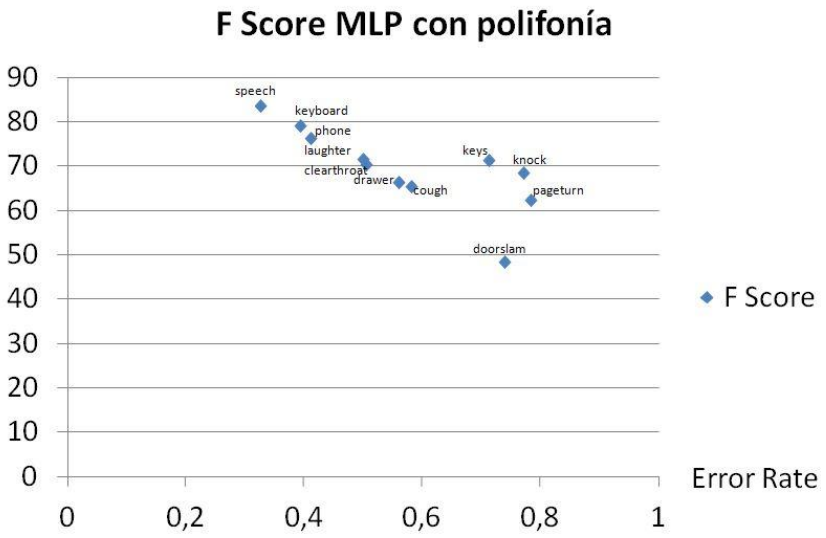

Figura 18. Gráfico de resultados por clase. MLP con polifonía, promedio F score= 71.00

Se repite la prueba cambiando el número de bandas para el cálculo del EMS.

Las condiciones utilizadas para el cálculo de EMS han sido:

- Bandas de sonido: 0 a 40 Hz, en intervalos de 2Hz
- Número de Filtros Gammatone: 40

Resultado con 20 bandas de sonido:

Resultado sin polifonía:

- Segmento: F-Score= 73.73%, ER: 0.4649

En este caso se consigue mejora en todas las clases salvo en Cough, Doorslam y Pageturn. Sin embargo no hay mejora en el resultado global.

Resultado con Polifonía:

- Segmento: F-Score= 66.14%, ER: 0.5718

Tampoco mejoran el resultado obtenido con 10 bandas.

Se repite la prueba manteniendo el número de bandas original, pero añadiendo ruido a los ficheros de referencia que forman parte del diccionario, según está especificado en el apartado 4.5.2 de este documento.

Resultado con ruido añadido:

Resultado sin polifonía:

- Segmento: F-Score= 68.68%, ER: 0.6440

Los resultados por clase de sonido (Segmento, F-ER) se muestran en la tabla de la figura 19:

Figura 19. Tabla de resultados por clase. NMF sin polifonía con ruido añadido

ER	F Score	Clase
0,1667	91,43	Clearthroat
0,2051	88,89	Cough
2,0333	49,59	Doorslam
0,303	82,14	Drawer
0,3958	80,41	Keyboard
0,8571	64	Keys
0,25	86,96	Knock
0,3333	80,95	Laughter
2,4889	9,68	Pageturn
0,3333	82	Phone
0,2745	84,78	Speech

Se observa en este caso que se obtiene mejora para todas las clases salvo para "Pageturn", donde se obtiene un pobre resultado de detección del 9.68%.

Sin embargo, se observa que si eliminamos "Pageturn" del cálculo del promedio, se obtiene en este caso el mejor resultado de detección de todas las alternativas estudiadas:

F score = 79.11%

Resultado con polifonía:

- Segmento: F-Score= 62.45%, ER: 0.6612

Si eliminamos Pageturn del cálculo del promedio, se obtiene en este caso F-score = 69.12%, algo inferior al 71% que se obtuvo sin ruido añadido. Deducimos que esta solución es más efectiva para la detección de sonidos monofónicos.

5.2.3 Comparación Rendimiento Python-Matlab para cálculo del EMS

A continuación se realiza una comparación de los tiempos de ejecución para el cálculo del EMS entre una implementación en MatLab y otra en Python.

Para realizar la comparación se calcula el EMS sobre un fichero wav de un segundo de duración en ambos entornos.

El tiempo total de la ejecución en MatLab fue de 2,81 segundos. Este tiempo se desglosa del siguiente modo:

- Lectura del fichero wav: 0,18 segundos
- Filtro Butterworth: 0,5 segundos
- Normalización de la señal: 0,062 segundos
- Banco de filtros Gammatone: 1,937 segundos
- Filtros paso bajo: 0,062 segundos
- Últimos cálculos de EMS: 0,06 segundos

Podemos ver que la mayor parte del tiempo se ocupa en el proceso del paso de la señal a través del banco de filtros Gammatone. Como se utilizan 40 filtros Gammatone, en promedio se utilizan 1,9375/40 = 0,0484 segundos por filtro. Este es el tiempo que emplea en promedio en cada llamada a la función “filter” de MatLab.

Con Python, el tiempo total de ejecución para el cálculo del EMS sobre el mismo fichero wav fue de 2,64 segundos.

Este tiempo se desglosa del siguiente modo:

- Lectura del fichero wav: 0,01 segundos
- Filtro Butterworth: 0,32 segundos
- Normalización de la señal: 0,22 segundos
- Banco de filtros Gammatone: 1,622 segundos
- Filtros paso bajo: 0.45 segundos
- Últimos cálculos de EMS: 0,02 segundos

En este caso, el promedio del tiempo de proceso por filtro Gammatone es de 1,622/40 = 0,040 segundos. Este es el tiempo que emplea en promedio en cada llamada a la función "lfilter" de Python.

En resumen, podemos concluir que el tiempo de procesado de EMS en Matlab y Python es similar. En este ejemplo, la mejora de Python respecto a MatLab fue de un 6% en tiempo de ejecución.

5.2.4 Detección en Tiempo real

Se realizan nuevas grabaciones de las siguientes clases de sonido: Clearthroat, Cough, Drawer, Keyboard, Keys, Phone, Speech, para generar un nuevo diccionario.

Se realiza una prueba que consiste en la reproducción secuencial (en vivo) de cada uno de los sonidos del diccionario con una duración aproximada de un segundo cada 10 segundos, para la detección monofónica. Para la detección polifónica se reproducen 2 sonidos simultáneamente durante un segundo cada 10 segundos (4 sonidos polifónicos en total en algo más de 40 segundos).

Se genera manualmente el fichero de anotación que corresponde a esta prueba y se utiliza el fichero generado por el módulo Python de detección para calcular el resultado con MatLab.

Resultado sin polifonía, en un intento de reproducción:

- Segmento: F-Score= 53.62%, ER: 1.6

No es difícil conseguir una detección donde aparezcan en el fichero de resultados todos los sonidos generados, aunque debido a los retardos en el proceso de detección los intervalos de finalización suelen tener un amplio margen de error.

Resultado con polifonía, en un intento de reproducción:

- Segmento: F-Score= 37.80%, ER: 1.8

Durante este intento, las clases Clearthroat y Drawer no fueron detectadas en sus intervalos de polifonía correspondientes. Se observa, que este comportamiento es bastante frecuente en otros intentos, puede ocurrir que en eventos simultáneos de sonido, uno de ellos sea enmascarado por otro que se reproduce simultáneamente con mayor potencia o duración.

6. CONCLUSIONES

EMS es un algoritmo adecuado para la extracción de características de una señal que pueden ser utilizadas para la detección de sonido, y su uso permite conseguir una mejora en la detección respecto a otros sistemas como VQT (F-score EMS con NMF = 68%, frente a F-Score VQT con NMF = 41%). La detección con EMS es mejor cuando se trata de sonido monofónico que en caso de polifonía.

Debido a que los sonidos proporcionados para entrenamiento eran sonidos sin ruido, y los proporcionados para la detección eran sonidos con diferentes niveles de ruido, la idea de añadir ruido a los ficheros de entrenamiento para que fueran más parecidos a los ficheros de detección introdujo mejoras en los resultados salvo para el caso de la clase 'Pageturn'. Eliminando esta clase en el cálculo del promedio de F-score, permitió un resultado de detección superior al 79% para el caso de NMF sin polifonía.

MLP es mejor algoritmo que NMF para la clasificación de sonidos. Una explicación puede ser que al ser EMS un algoritmo basado en el cálculo de la energía de bandas de frecuencia no se comporta de forma lineal en la superposición de sonidos. NMF sin embargo, está enfocado en la descomposición lineal de Matrices.

En general, el aumento del número de bandas de frecuencia para el cálculo de EMS produce mejoras en la detección. Sin embargo, una vez alcanzado un número de bandas de frecuencia representativo (en nuestro caso, 10 bandas de 0 a 30 Hz), el aumento de número de bandas para el cálculo del EMS ya no produce una mejora apreciable en la detección.

Python no es un lenguaje de programación que destaque por su velocidad de ejecución. Por ejemplo, para el cálculo de EMS, presenta un tiempo de procesado similar a MatLab. Sin embargo, permite la producción de código más legible y el uso de un amplio abanico de librerías disponibles.

Mediante el uso de las librerías de aprendizaje automático proporcionadas en Python (como scikit-learn) podemos hacer uso de otros algoritmos disponibles (MLP) y mejorar los resultados que fueron obtenidos utilizando NMF como algoritmo de detección en MatLab. En este proyecto se ha realizado una comparación utilizando dos algoritmos de clasificación, pero existe un amplio abanico de algoritmos disponibles que podrían mejorar estos resultados. A su vez, los algoritmos de clasificación permiten realizar ajustes adecuados a la aplicación que los utilice, por ejemplo, en el caso de MLP es posible ajustar el número de capas intermedias y el número de neuronas de cada capa. Sería posible realizar un estudio más profundizado sobre el ajuste de estos parámetros.

EMS permite la detección de sonidos en tiempo real, pero en los ejemplos utilizados en este proyecto sobre una plataforma Windows 10 en un ordenador de sobremesa se produce una demora en la detección de unos 3 segundos.

La detección en tiempo real es un área que ofrece posibilidades de mejora respecto a los resultados obtenidos en este proyecto. Es importante para obtener precisión en la determinación de los intervalos de tiempo donde se detecta el sonido, que la demora en la fase de caracterización debido al cálculo del EMS se reduzca drásticamente. Además

el uso de extensos diccionarios en aplicaciones reales puede llevar a la creación de modelos pesados, que contribuyan de forma importante al aumento del tiempo de respuesta en la fase de clasificación. La mejora global en el tiempo de detección puede conseguirse utilizando unidades de proceso más potentes, o con desarrollos de software que utilicen lenguajes de programación más eficientes y que permitan el procesamiento en paralelo. Es posible también reducir el número de filtros Gammatone que se utilizan para caracterizar las señales para que el cálculo del EMS sea más rápido, aunque en este caso a costa de perder información de la señal. Debe ponerse cuidado para que esta pérdida de información no contribuya a añadir errores en la clasificación del sonido.

7. REFERENCIAS

[1] «Siri – Apple (ES)» [En línea].
Available: https://www.apple.com/es/siri/
[Último acceso: 10 febrero 2020]

[2] «Apple (España)» [En línea].
Available: https://www.apple.com/es/
[Último acceso: 10 febrero 2020]

[3] C. Otero, «Alexa de Amazon: cómo funciona el asistente, qué hace y los modelos Echo que hay», 2018 [En línea]
Available: https://as.com/meristation/2018/10/31/betech/1541025737_394177.html
[Último acceso: 12 febrero 2020]

[4] «Amazon.es: compra online de electrónica, libros, deporte, hogar,…» [En línea].
Available: https://www.amazon.es/
[Último acceso: 12 febrero 2020]

[5] «Shazam» [En línea]
Available: https://www.shazam.com/es
[Último acceso: 14 febrero 2020]

[6] «Technology for a voice-enabled world | SoundHound Inc.» [En línea].
Available: https://www.soundhound.com/
[Último acceso: 14 febrero 2020]

[7] J. Laguarta, F. Hueto and B. Subirana, "COVID-19 Artificial Intelligence Diagnosis using only Cough Recordings," in IEEE Open Journal of Engineering in Medicine and Biology, doi: 10.1109/OJEMB.2020.3026928.

[8] J. M. Liss, S. LeGendre y A. J. Lotto, «Discriminating Dysarthria Type From Envelope Modulation Spectra,» Journal of Speech, Language, and Hearing Research, vol. 53, nº 5, pp. 1246-1255, 2014.

[9] «Scikit learn - sklearn.decomposition.NMF» [En línea]
Available:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html#
[Último acceso: 3 mayo 2020].

[10] «Scikit learn – sklearn.neural_network.MLPClassifier» [En línea]
Available:
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier
[Último acceso: 22 mayo 2020].

[11] «Python» [En línea].
Available: https://www.python.org/
[Último acceso: 1 febrero 2020]

[12] «DCASE 2016 Challenge - Task 2: Sound event detection in synthetic audio,» 2016. [En línea].
Available: http://dcase.community/challenge2016/task-sound-event-detection-in-synthetic-audio.
[Último acceso: 10 septiembre 2020].

[13] B. Blankertz, «The Constant Q Transform» [En línea]
Available: http://doc.ml.tu-berlin.de/bbci/material/publications/Bla_constQ.pdf
[Último acceso: 14 marzo 2020].

[14] D. Huang, M. Dong, H. Li, «A Real-Time Variable-Q Non-Stationary Gabor Transform for Pitch Shifting», 2015
Available: https://www.researchgate.net/publication/292361602_A_Real-Time_Variable-Q_Non-Stationary_Gabor_Transform_for_Pitch_Shifting.
[Último acceso: 14 marzo 2020].

[15] J. M. Gutierrez-Arriola, R. Fraile, C. A., D. T., J. J.L. y S. Mendoza, «Synthetic sound event detection based on MFCC,» Detection and Classification of Acoustic Scenes and Events 2016, Budapest, 2016.

[16] S. Dreiseitl y L. Ohno-Machado, «Logistic regression and artificial neural network classification models: a methodology review» [En línea]
Available: https://www.sciencedirect.com/science/article/pii/S1532046403000340
[Último acceso: 14 marzo 2020].

[17] J. A. Rodrigo, «Máquinas de Vector Soporte (Support Vector Machines, VMs)», 2017 [En línea]
Available:
https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines
[Último acceso: 7 marzo 2020].

[18] H. Phan, L. Hertel, M. Maass, P. Koch y A. Mertins, «CaR-FOREST: Joint Classification Regression Decision Forests for Overlapping Audio Event Detection,» arXiv:1607.02306, 2016.

[19] J. A. Rodrigo, «Árboles de decisión, Random Forest, Gradient Boosting y C5.0», 2017 [En línea]
Available:
cienciadedatos.net/documentos/33_arboles_de_prediccion_bagging_random_forest_boosting
[Último acceso: 10 marzo 2020].

[20] O. Harrison, «Machine Learning Basics with the K-Nearest Neighbors Algorithm», 2018 [En línea].
Available: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
[Último acceso: 21 marzo 2020].

[21] «Sound event detection in synthetic audio. Task results,» 2016. [En línea]. Available: http://www.cs.tut.fi/sgn/arg/dcase2016/task-results-sound-event-detection-in-synthetic-audio
[Último acceso: 18 septiembre 2020].

[22] A. Mesaros, T. Heittola y T. Virtanen, «Metrics for Polyphonic Sound Event Detection,» Applied Sciences, vol. 6, nº 6, p. 162, 2016.

[23] «PyCharm – IDE de Python para desarrolladores profesionales» [En línea]. Available: https://www.jetbrains.com/es-es/pycharm/
[Último acceso: 4 febrero 2020].

[24] «JetBrains: Essential tools for software developers and teams» [En línea]. Available: https://www.jetbrains.com/
[Último acceso: 4 febrero 2020].

[25] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. [En línea]
Available: https://scikit-learn.org/stable/
[Último acceso: 5 junio 2020].

[26] «SciPy» [En línea]
Available: https://docs.scipy.org/doc/scipy/reference/
[Último acceso: 8 febrero 2020].

[27] «NumPy – The fundamental package for scientific computing with Python» [En línea]
Available: https://numpy.org/
[Último acceso: 8 febrero 2020].

[28] «PyAudio» [En línea]
Available: https://pypi.org/project/PyAudio/
[Último acceso: 22 febrero 2020].

[29] «Matplotlib: Visualization with Python» [En línea]
Available: https://matplotlib.org/
[Último acceso: 19 marzo 2020].

[30] D. G. Manolakis, V. K. Ingle, Applied Digital Signal Processing: Theory and Practice, Cambridge University Press, New York (USA), 2011.

[31] R. Patterson, K. Robinson, J. Holdsworth, D. McKeown, C. Zhang, M. Allerhand, Complex sounds and auditory images, Auditory Physiology and Perception 83 (1992) 429–446.

[32] Cichocki, Andrzej, and P. H. A. N. Anh-Huy. “Fast local algorithms for large scale nonnegative matrix and tensor factorizations.” IEICE transactions on fundamentals of electronics, communications and computer sciences 92.3: 708-721, 2009

[33] Mesaros, A., Heittola, T., and Virtanen, T. "Metrics for polyphonic sound event detection", Applied Sciences, 6(6):162, 2016 PDF

[34] G. Lafay, E. Benetos y M. Lagrange, «Sound event detection in synthetic audio: Analysis of the {DCASE} 2016 task results,» 2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, pp. 11-15, 2017.

8. ANEXOS

ANEXO I: PRESUPUESTO

Para el cálculo del presupuesto, se ha utilizado un coste por hora de ingeniería de 40 €, que incluye gastos sociales e impuestos.

El coste total del proyecto ha sido dividido por las siguientes fases: Anteproyecto, Diseño, Desarrollo, Verificación, Corrección de Errores o Mantenimiento y Presentación de Resultados.

De igual modo, se han incluido los costes de equipos y alquileres necesarios.

El coste de licencias de software incluye el coste de la licencia de MatLab, pues los costes de Windows y las aplicaciones de Microsoft se han incluido en el coste del equipo de desarrollo, y el resto de licencias (Python, Pycharm) son gratuitas.

Fase del proyecto	Horas	Total
Anteproyecto	40	1600
Diseño	80	3200
Desarrollo	150	6000
Verificación	80	3200
Corrección de Errores	40	1600
Presentación	40	1600
		17200 €

Otros costes	Total
Ordenador portátil + Software Microsoft	600€
Micrófono	30€
Licencias de Software	800€
Alquiler de local y servicios	1000€
	2430 €

Coste Total del Proyecto	19630 €

ANEXO II: MANUAL DE USUARIO

2.1 Instalación

Instalación Python:

La versión instalada ha sido la 3.6.5. Se podrá descargar de la página https://www.python.org/downloads/release/python-365/

Instalación PyCharm:

(Versión Community) se puede descargar de la página: https://www.jetbrains.com/pycharm/download/#section=windows

2.2 Librerías importadas

Package	Version
Django	3.0.4
Pillow	7.0.0
PyAudio	0.2.11
asgiref	3.2.5
cycler	0.10.0
image	1.5.28
joblib	0.14.1
keyboard	0.13.4
kiwisolver	1.1.0
matplotlib	3.2.0
numpy	1.18.1
pip	9.0.3
pyparsing	2.4.6
python-dateutil	2.8.1
pytz	2019.3
scikit-learn	0.22.2.post1
scipy	1.4.1
setuptools	39.0.1
six	1.14.0
sqlparse	0.3.1

2.3 Descripción general de módulos Python creados

- **definitions.py:** Se especifican las variables de entrada y las constantes
- **commonFunctions.py:** Se definen funciones que son usadas por otros módulos
- **realTimeFunctions.py:** Se definen funciones que son usadas en la detección en tiempo real
- **split.py:** Parte los ficheros almacenados en la carpeta que se indica como argumento de entrada en ficheros de tamaño máximo 1 segundo, y los almacena en una carpeta de salida (también indicada como segundo argumento)
- **gammatone.py:** Retorna una señal en el dominio del tiempo tras pasarla por un filtro Gammatone con una determinada frecuencia central.
- **envelopeModulationSpectrum.py:** Retorna el EMS de un vector que se pasa como entrada. Para el cálculo del EMS es posible añadir un cierto nivel de ruido a los datos de entrada.
- **training.py:** Genera un fichero diccionario que contiene una matriz con los EMS de los ficheros contenidos en el directorio de training que se pasa como argumento.
- **eventDetectionNMF.py:** Genera el resultado de la detección en formato ficheros de texto, para el conjunto de entrada de ficheros almacenados en el directorio "path_input_files_to_detect" en base al diccionario contenido en "dictionary_file", según el algoritmo NMF.
- **eventDetectionNN.py:** Genera el resultado de la detección en formato ficheros de texto, para el conjunto de entrada de ficheros almacenados en el directorio "path_input_files_to_detect" en base al diccionario contenido en "dictionary_file", según el algoritmo MLP.
- **eventDetectionRealTime.py:** Genera el resultado de la detección en formato icono de imagen, y en formato ficheros de texto, para la entrada de audio tomada del micrófono en tiempo real, en base al diccionario contenido en "dictionary_file", según el algoritmo MLP.
- **generateDiccWithNoise.py**: Toma como entrada un conjunto de ficheros wav del directorio "pathBeforeSplit" y genera otro conjunto de ficheros añadiendo ruido de EBR=6dB a estos ficheros en el mismo path. Usado para la detección con ruido añadido.
- **generateNoise.py**: Genera un fichero wav con ruido Gaussiano. Este fichero será añadido a los ficheros de training para ser utilizado en la detección de la clase "noise".
- **recordFile.py**: Graba un fichero de audio tomado de la entrada de micrófono en "pathBeforeSplit_RT". Usado para generar el diccionario en la detección en tiempo real.

2.4 Detección con NMF

Parametros de Configuración

En primer lugar, en el fichero definitions.py se especificarán los parámetros de configuración:

GAMMATONE_CENTRAL_FREQS: lista de frecuencias centrales de los filtros Gammatone

vrModulationBands: Bandas de modulación en frecuencia

clases: tipos de sonidos a detectar

pathBeforeSplit: Ruta de los ficheros de audio antes de ser partidos.

pathAfterSplit: Ruta de los ficheros de audio después de ser partidos en trozos de un segundo de duración.

sampling_freq: frecuencia de muestreo

dictionary_file: Ruta y nombre de fichero donde se encuentra el diccionario

path_input_files_to_detect: Ruta de los ficheros de audio sobre los que habrá que detectar algunas de las clases que contendrán.

path_output_detected_files: Ruta de los ficheros de texto que, al procesar los ficheros de audio anteriores, contendrán los segundos en los que se ha detectado alguna de las clases.

Preparación de copia de ficheros de entrada

Copiar ficheros de training en *pathBeforeSplit.*

Copiar ficheros de detección en *path_input_files_to_detect.*

Generación de ficheros de sonido con sólo ruido

Ejecutar el módulo Python **generateNoise.py.**

Opción de añadir ruido a los ficheros de referencia

Si opcionalmente se desea añadir ficheros con ruido añadido en el diccionario, ejecutar el módulo Python: **generateDiccWithNoise.py**

Partición con solapamiento (split)

Ejecutar el módulo Python especificando el valor de los argumentos:

split.py pathBeforeSplit pathAfterSplit

Generación de diccionario

Ejecutar el módulo Python especificando el valor de los argumentos:

training.py pathAfterSplit

Detección

Ejecutar el módulo Python **eventDetectionNMF.py**.

Tomando como entrada el fichero diccionario y los ficheros de detección contenidos en *path_input_files_to_detect,* genera los ficheros de texto con el resultado en *path_output_detected_files.*

Evaluación del resultado con matlab

Ejecutar la siguiente función con MatLab (obtenida de la página de DCASE 2016):

Evaluate.

Parámetros de entrada:
sResultsFolder: ruta que contiene los ficheros *.txt* de salida de la detección.
sReferenceFolder: ruta que contiene los ficheros *.txt* de
anotaciones con los resultados exactos que deberían de ser obtenidos.

Parámetros de salida:
eSegmentBased: datos de los resultados basados en segmentos.
eEventBased: datos de los resultados basados en eventos.

2.5 Detección con modelo de Neural Networks

Parámetros de Configuración

En primer lugar, en el fichero definitions.py se especificarán los parámetros de configuración:

GAMMATONE_CENTRAL_FREQS: lista de frecuencias centrales de los filtros Gammatone

vrModulationBands: Bandas de modulación en frecuencia

clases: tipos de sonidos a detectar

pathBeforeSplit: Ruta de los ficheros de audio antes de ser partidos.

pathAfterSplit: Ruta de los ficheros de audio después de ser partidos en trozos de un segundo de duración.

sampling_freq: frecuencia de muestreo

dictionary_file: Ruta y nombre de fichero donde se encuentra el diccionario

path_input_files_to_detect: Ruta de los ficheros de audio sobre los que habrá que detectar algunas de las clases que contendrán.

path_output_detected_files: Ruta de los ficheros de texto que, al procesar los ficheros de audio anteriores, contendrán los segundos en los que se ha detectado alguna de las clases.

Preparación de copia de ficheros de entrada

Copiar ficheros de training en *pathBeforeSplit.*

Copiar ficheros de detección en *path_input_files_to_detect.*

Generación de ficheros de sonido con sólo ruido

Ejecutar el módulo Python **generateNoise.py.**

Opción de añadir ruido a los ficheros de referencia

Si opcionalmente se desea añadir ficheros con ruido añadido en el diccionario, ejecutar el módulo Python: **generateDiccWithNoise.py**

Partición con solapamiento (split)

Ejecutar el módulo Python especificando el valor de los argumentos:

split.py pathBeforeSplit pathAfterSplit

Generación de diccionario

Ejecutar el módulo Python especificando el valor de los argumentos:

training.py pathAfterSplit

Detección

Ejecutar el módulo Python **eventDetectionNN.py**.

Tomando como entrada el fichero diccionario y los ficheros de detección contenidos en *path_input_files_to_detect,* genera los ficheros de texto con el resultado en *path_output_detected_files.*

Evaluación del resultado con MatLab

Ejecutar la siguiente función con MatLab (obtenida de la página de DCASE 2016):

Evaluate.

Parámetros de entrada:
sResultsFolder: ruta que contiene los ficheros *.txt* de salida de la detección.
sReferenceFolder: ruta que contiene los ficheros *.txt* de anotaciones con los resultados exactos que deberían de ser obtenidos.

Parámetros de salida:
eSegmentBased: datos de los resultados basados en segmentos.
eEventBased: datos de los resultados basados en eventos.

2.6 Detección en tiempo real

Parámetros de Configuración

En primer lugar, en el fichero definitions.py se especificarán los parámetros de configuración:

clases_RT: tipos de sonidos a detectar

pathBeforeSplit_RT: Ruta de los ficheros de audio antes de ser partidos.

pathAfterSplit_RT: Ruta de los ficheros de audio después de ser partidos en trozos de un segundo de duración.

result_file_RT: nombre del fichero donde se almacenan los resultados

audio_samples_plotted_RT: Número de muestras de la señal de entrada que serán mostradas en un gráfico en la pantalla en tiempo real

lim_signal_in_plot_RT: Ajuste para el valor máximo de señal que aparecerá en el gráfico anterior

dictionary_file_RT: Ruta y nombre de fichero donde se encuentra el diccionario

min_signal_level: Al escuchar muestras de sonido del micrófono, minimo nivel de señal para considerar que se ha recibido sonido en vez de silencio

min_number_not_silent_samples: Al escuchar muestras de sonido del micrófono, mínimo número de muestras recibidas con sonido no silencioso para comenzar a calcular el EMS.

path_output_detected_files: Ruta de los ficheros de texto que, al procesar los ficheros de audio anteriores, contendrán los segundos en los que se ha detectado alguna de las clases.

Grabación de ficheros diccionario

Ejecutar el módulo **recordFile.py** para generar los ficheros de audio de referencia en *pathBeforeSplit_RT*.

Este módulo pide como parámetros de entrada nombre del fichero y segundos de grabación.

Partición con solapamiento (split)

Ejecutar el módulo Python especificando el valor de los argumentos:

split.py pathBeforeSplit_RT pathAfterSplit_RT

Generación de diccionario

Ejecutar el módulo Python especificando el valor de los argumentos:

training.py pathAfterSplit_RT

Detección

Ejecutar el módulo Python **realTimeDetection.py**.

Tomando como entrada el fichero diccionario y la entrada de audio del micrófono en tiempo real, genera iconos con el sonido detectado y los ficheros de texto con el resultado en *path_output_detected_files*.

Evaluación del resultado con MatLab

Ejecutar la siguiente función con MatLab (obtenida de la página de DCASE 2016):

Evaluate.

Parámetros de entrada:
sResultsFolder: ruta que contiene los ficheros *.txt* de salida de la detección.
sReferenceFolder: ruta que contiene los ficheros *.txt* de
anotaciones con los resultados exactos que deberían de ser obtenidos.

Parámetros de salida:
eSegmentBased: datos de los resultados basados en segmentos.
eEventBased: datos de los resultados basados en eventos.