DETECCIÓN DE TOSES EN UN CONJUNTO DESEQUILIBRADO

MINERÍA DE DATOS Y APRENDIZAJE AUTOMÁTICO, LENGUAJES Y SISTEMAS INFORMÁTICOS

BACHILLERATO DE INVESTIGACIÓN Y EXCELENCIA 2021

AUTORES:

BRUNOCORCUERA SÁCHEZ

PEDRO AYALA LÓPEZ

JORGE VALDIVIELSO BÁRCENA

TUTOR IES COMUNEROS DE CASTILLA:

UNIVERSIDAD
DE BURGOS

Generalidades

Título

Clasificador de sonidos

Área de investigación

Minería de datos y Aprendizaje automático, Lenguajes y Sistemas Informáticos.

Autores

Pedro Ayala López, Bruno Corcuera Sánchez y Jorge Valdivielso Bárcena

Tutores

José Francisco Díez Pastor. Área de Lenguajes y Sistemas Informáticos. Departamento de Ingeniería Informática. Universidad de Burgos

Carlos López Valcárcel . Departamento de Matemáticas. IES Comuneros de Castilla

Índice

1. Introducción

Esta memoria explica de manera detallada el proyecto que ha realizado un grupo de alumnos de segundo del bachillerato de investigación y excelencia del IES Comuneros de Castilla. El proyecto consistía originalmente en la clasificación e identificación de la intención comunicativa en sonidos de personas con parálisis cerebral, pero ante la falta de audios etiquetados de estas personas, finalmente el proyecto derivó a la diferenciación de una clase de audio de cualquier otra. El proyecto finalmente, consiste en diferenciar toses de otras 49 clases de audios diferentes con el repositorio ESC-50.

La dificultad de este proyecto estaba en conseguir que el programa identificará una clase minoritaria de un amplio conjunto de otras clases.

El problema de la detección de audios es muy útil pues puede diferenciar el tipo de audio de interés respecto a cualquier otro. Dependiendo del contexto puede ser interesante detectar roturas de ventanas o llantos de bebes en un sistema de vigilancia doméstico. Ronquidos en un sistema de monitorización de la calidad del sueño o cualquier otro tipo de aplicación que se nos ocurra. En este caso, poder detectar una tos, puede permitir advertir de una exposición peligrosa carga vírica de COVID-19.

2. Objetivos

El objetivo es lograr predecir una clase minoritaria sobre una mayoritaria en un conjunto de audios. Se decidió usar el repositorio *ESC-50* [1] En el cual hay diferentes audios ya etiquetados como sonidos de perros, gatos, estornudos… etc. En concreto se quiso conseguir un programa que fuera capaz de distinguir una clase previamente escogida, como es la tos, frente a las demás, lo que le añadía cierta dificultad ya que se trataba de conjuntos desequilibrados (aquellos en los que el número de ejemplos de una clase es muy superior al de las demás).

Para tal fin, es necesario el aprendizaje de Python como lenguaje de programación y de bibliotecas relacionadas con el *Deep Learning*, destacando de entre estas Scikit-learn, NumPy, Pandas y Tensorflow.

En último lugar, también figuran entre los objetivos principales de este trabajo la introducción en el campo de la investigación científica y el trabajo universitario, como parte de la realización de este proyecto dentro del Bachillerato de Investigación y Excelencia

[1] Proyecto colgado en GitHub. (https://github.com/karolpiczak/ESC-50)

3. Conceptos teóricos

A lo largo del proyecto ha sido necesario aprender a usar múltiples herramientas y para ello era importante conocer bien lo que eran exactamente. A continuación, se exponen algunos de los conceptos teóricos fundamentales.

3.1. Minería de datos

La minería de datos es un campo de la estadística y las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos. Para ello se emplean diversos métodos, entre los que se encuentran la inteligencia artificial, el aprendizaje automático, la estadística y los sistemas de bases de datos. [1]

El objetivo final del proceso de minería de datos es obtener información de un conjunto de datos, (patrones o relaciones) analizarla y transformarla en una estructura comprensible para su uso posterior. En este proyecto, por ejemplo, lo que se busca es un patrón que identifique la tos y lo sepa diferenciar de otros sonidos.

El objetivo de la minería de datos es el de encontrar patrones en conjuntos de datos. Está estrechamente relacionado con el término “extracción de conocimientos en bases de datos”, del inglés *knowledge discovery in databases* o (KDD).

3.2. Aprendizaje automático (*Machine Learning*)

El aprendizaje automático o aprendizaje de máquinas (*machine learning*) es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial que sirve para desarrollar técnicas que permitan que los ordenadores y otros dispositivos aprendan por sí solos. Que una máquina aprenda quiere decir que mejora con la experiencia, es decir, que cada vez funciona de una manera más precisa y exacta sin que esa información formará parte de ella al principio.

Los modelos o programas resultantes deben ser capaces de generalizar comportamientos e inferencias para un conjunto más amplio de datos (pudiendo en la teoría llegar este al infinito). Es decir, que sea capaz de ofrecer resultados en casos nunca antes vistos por el modelo. Muchos problemas son de clase *NP-hard*, por lo que gran parte de la investigación realizada en el aprendizaje automático está enfocada al diseño de soluciones factibles a esos problemas. [2]

El Aprendizaje Automático es utilizado en una gran variedad de ámbitos, desde el reconocimiento de voz, hasta la Exploración o Minería de datos, existiendo una gran variedad de algoritmos. Este tipo de algoritmos tienen dos fases fundamentales: la fase de entrenamiento y la fase de evaluación.

La fase de entrenamiento consiste principalmente en proporcionar observaciones o ejemplos al sistema, de los cuales este debe aprender. En este punto surgen dos alternativas:

- Aprendizaje Automático Supervisado (Supervised Machine Learning).
- Aprendizaje Automático No Supervisado (Unsupervised Machine Learning).

El primero consiste en proporcionar al sistema un conjunto de datos o ejemplos de aprendizaje etiquetados, es decir, contienen la respuesta correcta. Sin embargo, en el método no supervisado, los datos de entrenamiento no están etiquetados, dando lugar a

algoritmos de agrupamiento o *clustering* que permitan descubrir nuevos patrones de clasificación de un cierto conjunto de observaciones.

La fase de evaluación es llevada a cabo una vez el modelo ha sido entrenado. Esta fase implica proporcionar al sistema un subconjunto de datos o ejemplos, que el propio sistema debe etiquetar. Conociendo las verdaderas etiquetas de los datos proporcionados sería posible verificar los resultados de la evaluación, caracterizando así al sistema en función de sus aciertos y fallos.

El aprendizaje automático tiene una amplia gama de aplicaciones, incluyendo motores de búsqueda, diagnósticos médicos, detección de fraude en el uso de tarjetas de crédito, análisis del mercado de valores, clasificación de secuencias de ADN, reconocimiento del habla y del lenguaje escrito, juegos y robótica.

3.3. Clasificación

La clasificación es una parte del aprendizaje supervisado que consiste en predecir a qué clase pertenece una información. Aunque pueda parecer que recoge todo el aprendizaje supervisado no es así ya que se obtienen solamente categorías, no datos numéricos.

Para que un programa pueda predecir adecuadamente es preciso ayudarle diciéndole de qué tipo son ciertas informaciones, de esta manera, después tendrá una base que le permitirá predecir la categoría de una futura información que no conozca.

Los clasificadores más utilizados son las redes neuronales, las máquinas de vectores de soporte, el algoritmo de: *K* vecinos más cercanos, los modelos de mixturas, el clasificador bayesiano ingenuo, los árboles de decisión y las funciones de base radial.[3]

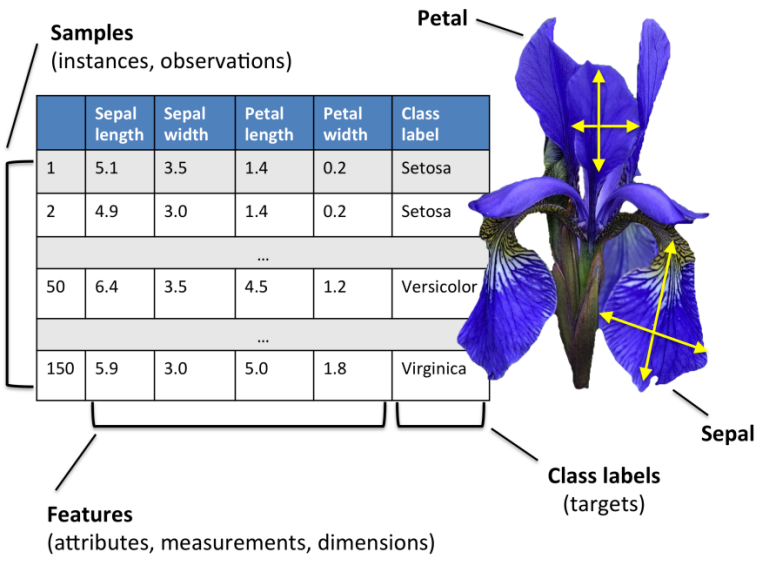
3.4. Conjunto de datos

Un conjunto de datos o *dataset* es el contenido de una única tabla de datos o matriz de datos estadística y está formada por ejemplos (filas), cada uno de los cuales tiene varios atributos y una clase. [4]

Un conjunto de datos contiene los valores para cada una de las variables que corresponden a cada miembro del conjunto de datos. Cada uno de estos valores se conoce con el nombre de dato. El conjunto de datos puede incluir datos para uno o más miembros en función de su número de filas.

Los conjuntos de datos tan grandes que aplicaciones tradicionales de procesamiento de datos no los pueden tratar se llaman *Big Data*.

Los datos pueden ser de diferentes tipos, los más comunes son los siguientes: números naturales, números enteros, números de coma flotante (decimales), cadenas alfanuméricas, estados (booleanos)...etc.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Fig. 1 Ejemplo de una tabla de conjunto de datos (Dataset Iris Versicolor)

3.5. Conjunto desequilibrado

Los conjuntos de datos desequilibrados son aquellos conjuntos en los hay muchos más datos de unas clases que de otras.

Los clasificadores tradicionales tratan de maximizar la tasa de acierto, cosa que en este caso no tiene utilidad, ya que un clasificador en un conjunto de datos en los que la clase minoritaria supone un 1% del conjunto, puede dar una tasa de acierto del 99% diciendo que todos los datos pertenecen a la clase mayoritaria. En este los algoritmos así no sirven ya que no ha identificado bien ninguna clase minoritaria, que son las que interesan en estos conjuntos. [5]

Un ejemplo común de conjunto de datos desequilibrados es la detección de fraudes con tarjeta de crédito, donde los puntos de datos para fraude (1), generalmente son muy menores en comparación con los de no fraude (0).

En estos conjuntos de datos, la clase mayoritaria se etiqueta como negativa (0), y la clase minoritaria es positiva (1).

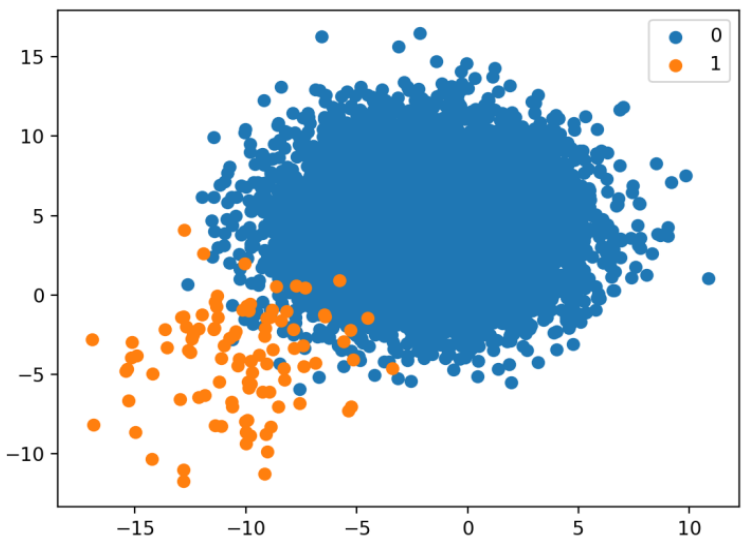


Fig. 2 Ejemplo gráfico de un conjunto desequilibrado

Como la tasa de acierto en un conjunto desequilibrado no resulta útil, se emplean estas métricas:

Recall (Sensibilidad): ¿Del total de instancias positivas cuantas han sido detectadas? Esta medida es interesante cuando clasificar un positivo como negativo es muy perjudicial. Ejemplo, en un test médico es importante que se tenga una alta sensibilidad. Un test con 100% de sensibilidad detectará el 100% de los pacientes con una determinada enfermedad. No se le escapa ningún positivo real.

$$Recall = \frac{TP}{TP + FN} = \frac{Positivos\_predichos\_correctamente}{Positivos\_Reales\_Totales}$$

Precisión (Especificidad): ¿Del total de predicciones positivas cuantas en realidad lo eran? Esta medida es interesante por ejemplo en un sistema de recuperación de la información, si quieres recuperar documentos o fotos clasificadas de una determinada manera es más importante no tener falsos positivos que recuperar absolutamente todos los casos. No queremos falsos positivos, no pasa nada si no se recuperan todos los positivos reales. (En la prensa, cuando hablan de precisión no se refieren a precisión sino a tasa de acierto).

$$Precision = \frac{TP}{TP + FP} = \frac{Positivos\_predichos\_correctamente}{Positivos\_Predichos}$$

F-Measure (Medida F): es una medida que penaliza cuando una de las dos medidas anteriores es baja, por ello es la medida que hay que usar para saber si un clasificador funciona o no con un conjunto desequilibrado. [5]

$$F.Measure(F1) = 2x\frac{RecallxPrecision}{Recall + Precision}$$

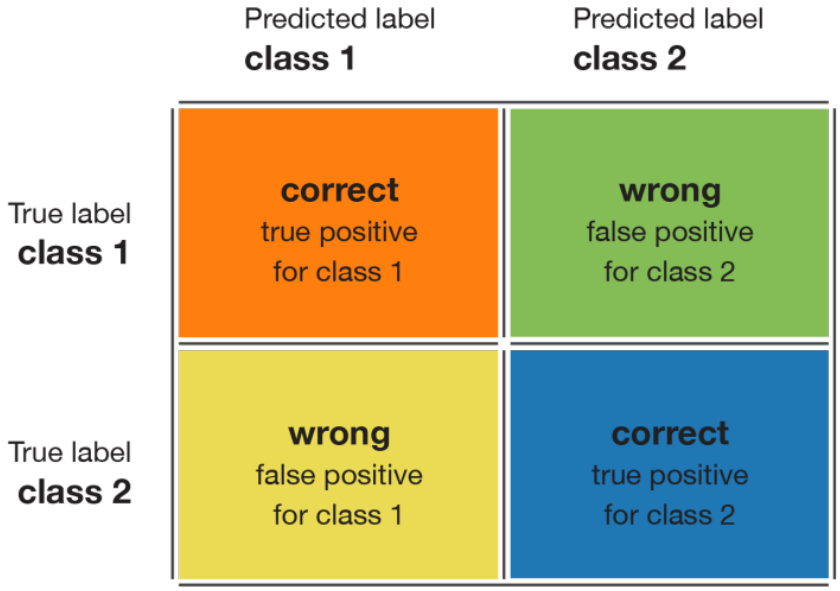


Fig. 3 Tabla que representa de forma gráfica una matriz de confusión con un conjunto desequilibrado.

Para el procesamiento de datos de este tipo de conjuntos se puede realizar dos técnicas diferentes las cuales tratan de igualar el número de instancias que hay de las diferentes clases. Estos métodos son:

Eliminar ejemplos de la mayoritaria:

Random Undersampling: elimina instancias de la clase mayoritaria de manera aleatoria para conseguir un conjunto de datos equilibrado.

Añadir ejemplos de la minoritaria:

Random Oversampling: crea copias de las instancias de la minoritaria que ya están presentes en el conjunto de datos. [6]

SMOTE: el método *SMOTE* (*Synthetic Minority Over-sampling Technique*) crea mediante interpolación, ejemplos artificiales que no son exactamente iguales. Así el clasificador tendrá más capacidad de generalización y podrá reconocer ejemplos de la clase minoritaria que no sean exactamente igual a los del conjunto de entrenamiento, lo que lo hace más efectivo.

El funcionamiento es el siguiente: de cada instancia de la clase minoritaria se buscan un número de vecinos (k). Se selecciona uno de ellos aleatoriamente y se crea una instancia artificial en un punto aleatorio de la línea que une ambas instancias.

Borderline SMOTE: es una variante que en lugar de crear instancias artificiales a partir de todas y cada una de las instancias de la clase minoritaria se crean instancias artificiales solo de las instancias que están en la frontera entre la clase minoritaria y la mayoritaria. [7]

3.6. Validación Cruzada (Cross Validation)

La validación cruzada o *k-fold Cross Validation* consiste en tomar el conjunto de datos originales y dividirlo en *k* subconjuntos (*folds*). En el momento de realizar el entrenamiento, se toma cada *k* subconjunto como conjunto de prueba del modelo, mientras que el resto de los datos se toma como conjunto de entrenamiento.

Este proceso se repite *k* veces, y en cada iteración se selecciona un conjunto de prueba diferente, mientras los que los datos restantes se emplean, como ya se ha mencionado, como conjunto de entrenamiento. Una vez finalizadas las iteraciones, se calcula la precisión y el error para cada uno de los modelos producidos, y para obtener la precisión y el error final se calcula el promedio de los *k* modelos entrenados. [10]

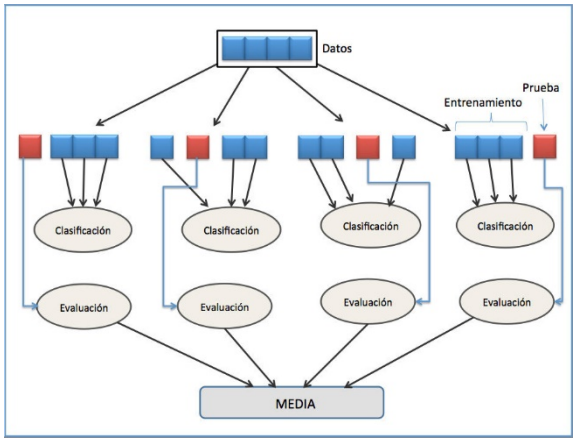


Fig. 4 Gráfica que muestra la división de un conjunto de datos para la validación cruzada

3.7. Regresión logística

El algoritmo de regresión logística es uno de los más utilizados actualmente en aprendizaje automático. Siendo su principal aplicación los problemas de clasificación binaria. Es un algoritmo simple en el que se pueden interpretar fácilmente los resultados obtenidos e identificar por qué se obtiene un resultado u otro

La Regresión Logística describe y estima la relación entre una variable binaria dependiente y las variables independientes.

A la función que relaciona la variable dependiente con las independientes también se le llama función sigmoidea. La función sigmoidea es una curva en forma de S que puede tomar cualquier valor entre 0 y 1, pero nunca valores fuera de estos límites. La ecuación que define la función sigmoidea es:

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

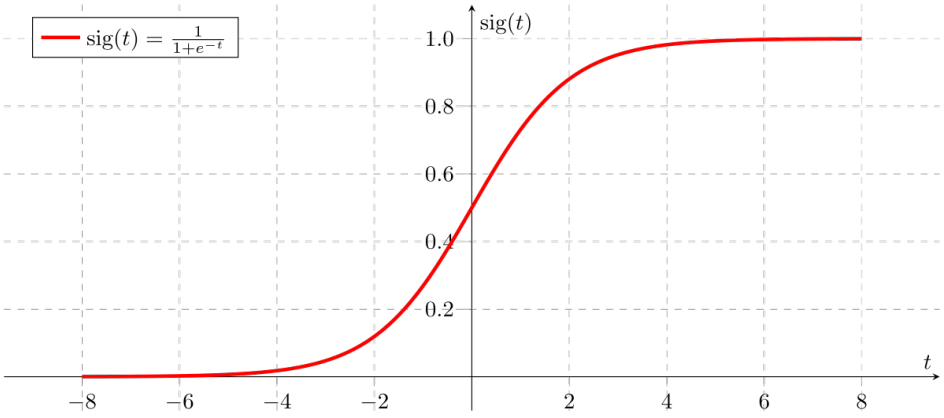


Fig. 5 Representación gráfica de una función logística

Si la curva va a infinito positivo la predicción se convertirá en 1, y si la curva pasa el infinito negativo, la predicción se convertirá en 0. Si la salida de la función Sigmoide es mayor que 0.5, podemos clasificar el resultado como 1 o SI, y si es menor que 0.5 podemos clasificarlo como 0 o NO. Por su parte si el resultado es 0.75, podemos decir en términos de probabilidad como, hay un 75% de probabilidades de que sea positivo.

De esta manera, se puede utilizar para dividir un conjunto de datos de dos clases y poder predecir un nuevo caso según en la posición en la que se encuentre.

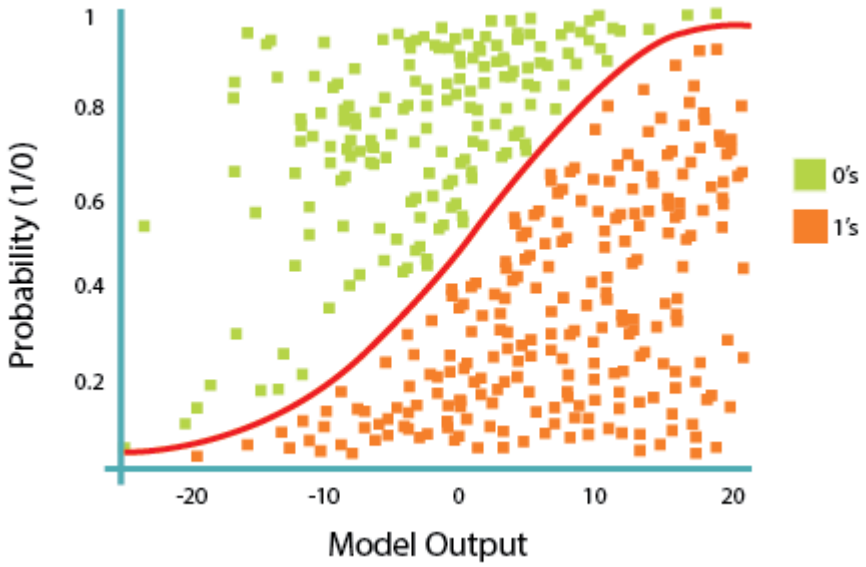


Fig. 6 Ejemplo de clasificación binaria con Regresión Logística

3.8. Algoritmos de Boosting

El *boosting* consiste en combinar los resultados de varios clasificadores débiles para obtener un clasificador robusto. Cuando se añaden estos clasificadores débiles, se hace de modo que estos tengan diferente peso en función de la exactitud de sus predicciones. Luego de que se añade un clasificador débil, los datos cambian su estructura de pesos: los casos que son mal clasificados ganan peso y los que son clasificados correctamente pierden peso. Así, los clasificadores robustos se centran de mayor manera en los casos que fueron mal clasificados por los clasificadores débiles.

En este caso los clasificadores de *boosting* que se han utilizado para el clasificador robusto han sido *GradientBoostingClassifier* y *XGBoost* [9]

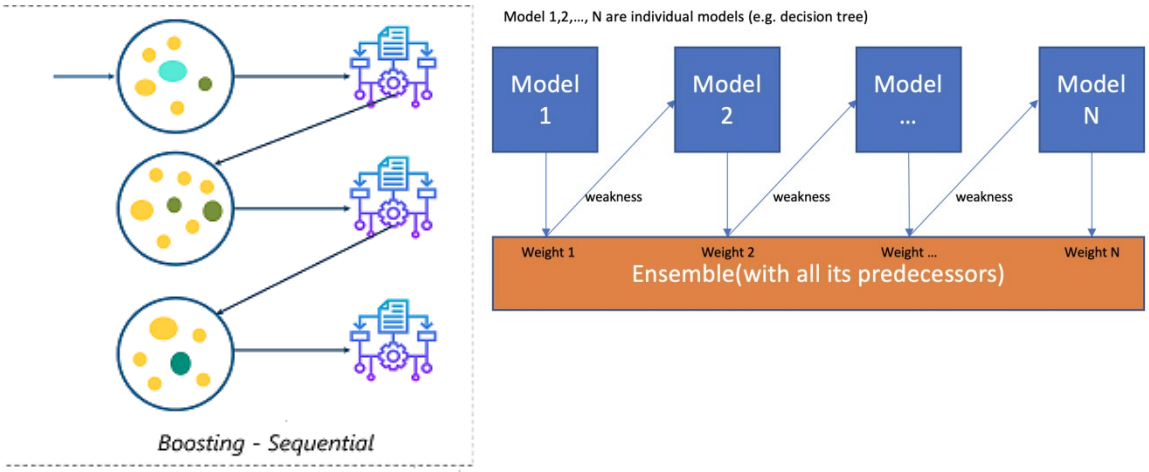


Fig. 7 Muestra el proceso de combinación de clasificadores en los algoritmos de boosting.

3.9. VotingClassifier

El *Voting* es una de las formas más sencillas de combinar las predicciones de múltiples algoritmos de aprendizaje automático. El clasificador por votación no es un clasificador propiamente dicho, sino una envoltura para un conjunto de diferentes clasificadores que se entrenan y valoran en paralelo con el fin de explotar las diferentes peculiaridades de cada algoritmo.

Podemos entrenar un conjunto de datos utilizando diferentes algoritmos y combinarlos para predecir el resultado final. El resultado final de una predicción se toma por mayoría de votos según dos estrategias diferentes:

Hard voting / Majority voting: La votación dura es el caso más sencillo de la votación por mayoría. En este caso, se elegirá la clase que haya recibido el mayor número de votos Nc(yt). Aquí predecimos la etiqueta de clase y^ mediante la votación por mayoría de cada clasificador. [11]

4. Herramientas utilizadas

Para poder realizar correctamente el proyecto fue preciso recurrir a diferentes herramientas, estas incluyen un lenguaje de programación, un repositorio de audios, dos redes neuronales y diferentes bibliotecas.

4.1. Google Collaboratory

Google Collaboratory es un entorno gratuito de *Jupyter Notebook* que no requiere configuración, que se ejecuta completamente en la nube y que emplea el lenguaje de Python.

La razón por la que se escogió este entorno para trabajar, además de por su gratuidad, es porque ofrece la posibilidad de trabajar en un mismo código al mismo tiempo desde varios dispositivos, lo que resulta cómodo y agradable.

Fig. 8 Logotipo de Python y Google Colab

4.2. Python

Python es un lenguaje de programación creado en 1991 por el informático Guido van Rossum que está basado en un lenguaje previo denominado ABC.

CPython es la referencia de implementación del lenguaje Python, es completamente gratuita y de código abierto así como todos los módulos o bibliotecas de la plataforma.

Cabe destacar que Python es un lenguaje simple de aprender y utilizar. Sin embargo, su simplicidad no implica falta de funcionalidad, es un lenguaje que se ha extendido a todos los ámbitos de la informática y ha sido mejorado por los usuarios.

Esto permite hacer prácticamente cualquier cosa con él sin mucho esfuerzo, al contrario que con lenguajes de programación preexistentes, como C++, Java o FORTRAN, que son por naturaleza mucho más complejos. [12]

Fig. 9 Logotipo de Python

4.3. ESC-50

ESC-50 (*Dataset for Environmental Sound Classification*) es una colección de grabaciones ambientales breves disponibles en un formato unificado. Son 2000 audios de 5 segundos de duración, 44’1 kHz, canal único y *Ogg Vorbis* comprimido a 192 kbit/s de 50 clases diferentes (hay 40 audios por clase). Todos los clips se pueden encontrar en un repositorio de GitHub [13] a través del proyecto Freesound.org.

Para la realización del proyecto se ha usado este repositorio ya que ofrecía una gran cantidad de audios variados de gran calidad clasificados por clases. A demás, es una fuente fiable, lo que significa que no es necesario verificar que todos y cada uno de los audios estén correctos.

4.4. VGGish

VGGish es una red neuronal convolucional desarrollada por Google que está pre-entrenada con la base de datos YouTube-8M y que permite obtener 128 características por cada segundo de audio. YouTube-8M es un archivo a gran escala de audios etiquetados formado por millones de videos de YouTube con diverso vocabulario y con anotaciones de gran calidad generadas por ordenador. [14]

Esta red neuronal se encuentra dentro de TensorFlow Models, que es un repositorio donde se encuentran muchos modelos de aprendizaje automático.

4.5. YAMNet

YAMNet es una red profunda que emplea la arquitectura de convolución separable en profundidad Mobilenet_v1 y que predice 521 clases de eventos de audio del *corpus AudioSet-YouTube* en el que se entrenó.

Esta red permite sacar 1024 características por cada décima parte de segundo de audio. Lo que hace que sea mucho más precisa, aunque esto también ralentice el proceso.[15]

Esta red neuronal se encuentra dentro de *TensorFlow Models*, repositorio en donde se encuentran también otros muchos modelos de aprendizaje automático.

4.6. Bibliotecas

Las bibliotecas, más conocidas como librerías, son un conjunto de elementos funcionales codificados en el lenguaje de programación más adecuado para su función concreta que no funcionan de manera autónoma, sino que son utilizadas por otros programas como una herramienta más. Una biblioteca puede depender de otras.

4.6.1. Scikit-learn

Scikit-learn [2] es una biblioteca de aprendizaje automático de código abierto. Esta puede ser utilizada tanto para aprendizaje supervisado como para no supervisado (*supervised learning* y *unsupervised learning*), respectivamente. También proporciona múltiples herramientas para el entrenamiento de modelos, procesamiento de datos, selección y evaluación de modelos, además de otras muchas utilidades.

---

[2] (https://scikit-learn.org/stable/)

Scikit-learn proporciona decenas algoritmos y modelos de aprendizaje automático preconstruidos, llamados "*estimators*". Cada uno de estos "*estimators*" puede ser ajustado para un conjunto de datos utilizando su respectivo método de ajuste (*fit method*). [16]

Fig. 10 Logotipo de Scikit-learn

4.6.2. NumPy

NumPy [3] es una biblioteca de software libre bajo licencia BSD imprescindible para Python que permite crear vectores y matrices multidimensionales. A demás, contiene una gran variedad de funciones y herramientas que permiten integrar distintos tipos de código.

También puede ser empleado para almacenar de manera multidimensional datos genéricos y puede definir tipos arbitrarios de datos, lo que permiten a NumPy integrarse de manera eficaz con otras muchas bases de datos. [17]

Fig. 11 Logotipo de Numpy

[3] (https://numpy.org/)

4.6.3. Pandas

Pandas [4] es una biblioteca que funciona como una extensión de NumPy y cuyas funciones son manipular y analizar datos en Python. Sus principales características incluyen las siguientes: procesar una gran variedad de conjuntos de datos en diferentes formatos, facilitar la descarga/importación de datos de múltiples fuentes (como CSV o bases de datos como *SQL*) e integrarse bien con otras librerías de Python como *statsmodels, SciPy y Scikit-Learn*. [18]

Fig. 12 Logotipo de pandas

4.6.4. Imbalanced-learn

Imbalanced-learn [5] es un paquete de Python que forma parte de los proyectos *scikit-learn-contrib* (por lo que es compatible con *scikit-learn*) que ofrece una serie de técnicas de remuestreo comúnmente utilizadas en conjuntos de datos desequilibrados. [19]

En esta librería, se pueden encontrar modelos de las diferentes técnicas principales de remuestreo que existen:

[4] (https://pandas.pydata.org/)

[5] (https://imbalanced-learn.org/stable/)

-Submuestreo de la(s) clase(s) mayoritarias.

-Sobremuestreo de la clase minoritaria.

-Combinación de sobremuestreo y submuestreo.

-Creación de conjuntos de conjuntos equilibrados.

Fig. 13 Logotipo de los proyectos: scikit-learn-contrib

4.6.5. TensorFlow

TensorFlow [6] es una biblioteca de código abierto para el aprendizaje automático que consiste en un almacén de librerías para la comunidad de investigadores que sirve para impulsar los desarrollos del *machine learning*. [20]

Entre sus ventajas destaca que se puede utilizar como un simple compilador de proyectos o para publicar e innovar en tus proyectos.

Fig. 14 Logotipo de Tensor Flow

[6] (https://www.tensorflow.org/)

5. Desarrollo del proyecto

Para aumentar nuestro conocimiento sobre el uso de las diferentes herramientas necesarias para la realización del proyecto se realizaron diferentes prácticas, las cuales, nos permitieron abarcar diferentes elementos. Estas prácticas abarcan desde el estudio de conceptos teóricos generales hasta el uso de funciones específicas para nuestro proyecto. A continuación, se explican las prácticas que se llevaron a cabo.

5.1. Práctica 1 (Introducción a la clasificación)

En esta primera práctica se estudiaron conceptos teóricos como "minería de datos", "aprendizaje automático" y "aprendizaje supervisado" entre otros. Además de hacer un planteamiento general del proyecto.

Se introdujo en el uso de librerías necesarias para el posterior manejo de los datos como es Pandas y NumPy, el funcionamiento de un clasificador sencillo como *K* vecinos más cercanos y Árboles de decisiones.

Se empezó dividiendo el conjunto de audio ESC-50 en 5 particiones predeterminadas(*folds*), obteniendo los conjuntos para entrenar y para testear el clasificador para la posterior validación cruzada. Cada conjunto de estos, tenía 1600 audios para entrenamiento y 400 audios para test.

```
atributos = list(map(lambda x:"Att"+str(x),range(2048)))
metadatos_df = pd.read_csv("ESC-50/meta/esc50.csv")
clase = Caracteristicas_df.Clase
Fold =  {}
last = metadatos_df.fold.iloc [-1]+1
#Tuplas con  Test y Train
for i in range(1,last):

  df_test = Caracteristicas_df [Caracteristicas_df.fold == i]
  df_train =Caracteristicas_df [Caracteristicas_df.fold != i]
```

```
X_train = df_train[atributos].values
X_test = df_test[atributos].values

y_train = df_train.Clase.values
y_test = df_test.Clase.values

Fold["Test" + str(i)] = (X_test,y_test)
Fold["Train" + str(i)] = (X_train,y_train)
```

El conjunto se separó en dos clases, tos (1) y no tos (0), para convertirlo en un conjunto desequilibrado.

	filename	fold	target	category	esc10	src_file	take	Clase
0	1-100032-A-0.wav	1	0	dog	True	100032	A	0
1	1-100038-A-14.wav	1	14	chirping_birds	False	100038	A	0
2	1-100210-A-36.wav	1	36	vacuum_cleaner	False	100210	A	0
3	1-100210-B-36.wav	1	36	vacuum_cleaner	False	100210	B	0
4	1-101296-A-19.wav	1	19	thunderstorm	False	101296	A	0
...	...	...	...	...	...	...	...	...
1995	5-263831-B-6.wav	5	6	hen	False	263831	B	0
1996	5-263902-A-36.wav	5	36	vacuum_cleaner	False	263902	A	0
1997	5-51149-A-25.wav	5	25	footsteps	False	51149	A	0
1998	5-61635-A-8.wav	5	8	sheep	False	61635	A	0
1999	5-9032-A-0.wav	5	0	dog	True	9032	A	0

2000 rows × 8 columns

Fig. 15 *Dataset* de ESC-50 con la clase de “tos” y “no tos” añadida

5.2. Práctica 2 (Introducción a VGGish y YAMNet)

Con esta práctica se aprendió lo que eran las redes neuronales *VGGish* y *YAMNet*. Se usaron estas redes para adaptarlas a nuestras necesidades y obtener las características de un audio cualquiera del repositorio ESC-50 y a continuación se aplicaron estos conocimientos y los de la práctica anterior para obtener una tabla con las características de todos los audios del repositorio ESC-50. Estas características obtenidas, en *VGGish* se obtenían 128 características por cada segundo de audio(5x128) y en *YAMNet*, 1024 por cada decisegundo del audio (50x1024). En ambos casos, se optó por realizar la media de todos los segundos y se añadía también, la varianza de todos los segundos, obteniendo así 256 características en *VGGish* y 2048 en *YAMNet* por audio.

Ya que este es un proceso largo, se convirtieron todas las características de los audios en un archivo (.csv) para poder descargarlo e importarlo en las siguientes prácticas.

filename	Att0	Att1	Att2	Att3	Att4	Att5	Att6	Att7	Att8	Att9	Att10	Att11	Att12	
1-100032-A-0.wav	1.779540	0.403408	0.577516	0.064785	2.924030	2.498055	0.0	2.081190	2.686362	1.240811	2.100662	3.65069	0.131920	2.0
1-100038-A-14.wav	0.000000	0.148130	0.912608	0.012334	0.000000	0.000000	0.0	0.000000	0.000000	0.012466	0.000000	0.00000	0.001692	0.0
1-100210-A-36.wav	0.000000	0.000000	0.113216	1.652952	0.000000	0.000000	0.0	0.003575	0.000000	0.000000	0.000000	0.00000	0.000000	0.0
1-100210-B-36.wav	0.000000	0.008155	0.111582	1.656614	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.0
1-101296-A-19.wav	0.000841	0.039995	0.026224	0.261450	0.000745	0.000000	0.0	0.000814	0.000000	0.000000	0.000000	0.00000	0.022463	0.0

5 rows × 2050 columns

Fig. 16 Tabla obtenida con los audios en las filas y las características en las columnas

5.3. Práctica 3 (Introducción a los algoritmos de clasificación)

En esta segunda práctica se estudiaron los conceptos de, *recall*, especificidad y *F-Measure* y las diferentes técnicas para equilibrar el conjunto de audios entre la clase minoritaria (tos) y la clase mayoritaria (tos) con *Random Undersampling, Random Oversampling, SMOTE y Borderline SMOTE*.

Primero entrenamos el clasificador *K* vecinos más cercanos, Regresión Logística y Clasificadores de Árboles de decisiones con las características que se obtuvieron de VGGish.

Los resultados fueron bastante malos, por lo que probamos distintos clasificadores. Con YAMNet los resultados fueron muy superiores, por lo que continuamos optimizando los clasificadores probando algunos más avanzados. Finalmente, para intentar obtener los mejores resultados, utilizamos Voting Classifier. Consiste en emplear los clasificadores que mejor funcionan para crear un clasificador superior que se basa en tomar por válido lo que la mayoría de los clasificadores que se utilizan de base predicen. Con Voting, utilizando como clasificadores base *XGBoost*, Regresión Logística y *Gradient Boosting* y con *Borderline SMOTE*, se consiguieron unos muy buenos resultados.

```
BorderlineSMOTE
accuracy: 0.9825000000000002
precision: 0.5897619047619048
recall: 0.625
f1: 0.584863339275104
```

Fig. 17 Resultados del clasificador Voting con (XGBoost, Regresión Logística y Gradient Boosting y Borderline SMOTE)

La matriz de confusión del *Voting* con *BordelineSMOTE* muestra que, de los 400 audios de test, 391 positivos acertó, acertó 3 negativos, dio 5 falsos positivos y solo un falso negativo.

```
[[391   1]
 [  5   3]]
```

Fig. 18 Matríz de confusión

La conclusión fue que *YAMNet*, al extraer una cantidad muy superior de características y no solo por cada segundo, sino que, por cada decisegundo, se obtiene una información más precisa del audio, que permite al clasificador obtener mejores resultados.

Los resultados con los distintos clasificadores se fueron añadiendo a un Excel con la medida *f-measure.*

VGGish				
	Under Sampling	Over Sampling	SMOTE	BorderlineSMOTE
K vecinos	0,065	0,11	0,062	0,072
Tree	0,137	0,309	0,23	0,208
Regr. Logist.	0,096	0,254	0,256	0,253
Random Forest	0,345	0,251	0,35	0,185

YAMNet				
	Under Sampling	Over Sampling	SMOTE	BorderlineSMOTE
K vecinos	0,1	0,199	0,183	0,224
Tree	0,163	0,293	0,265	0,281
Regr. Logist.	0,288	0,523	0,515	0,474
Random Forest	0,265	0,252	0,334	0,243
SVM	0,113	0,28	0,234	0,2
Neural Networks	0,172	0,085	0,389	0,216
GradientBoosting	0,183	0,501	0,482	0,513
XGBoost	0,284	0,527	0,57	0,49
AdaBoost	0,27	0,492	0,5	0,476
Voting(XGBoost, GradientBoosting,Regr. Logist)	0,3	0,522	0,561	0,584
Bagging	0,236	0,474	0,484	0,42
StandardScaler	0,185	0,46	0,434	0,449

Fig. 19 Tabla con los resultados de las distintas técnicas de clasificación probadas

5.4. Práctica 4 (Creación de un programa funcional)

El objetivo de esta práctica es crear un programa demo, que permita procesar un audio más largo, en este caso 30 segundo, y con el clasificador previamente guardado en un pipeline, procesar este audio con una "ventana deslizante" de medio segundo. A pesar de que en el conjunto ESC-50 los audios son de 5 segundos, una tos ocupa un espacio de tiempo muy reducido, por lo que, si se realiza con cinco segundos, al procesar el audio, la mayor parte de ese tiempo contiene sonidos que no son una tos, lo que dificulta enormemente la identificación de estas toses en el audio. Para solucionarlo se optó por procesar el audio con periodos de tiempo más cortos, en concreto medio segundo. Después, se obtiene una gráfica en la que se pueda ver dónde y con qué probabilidad detecta toses a lo largo del mismo.

El primer paso es guardar el clasificador para utilizarlo en la demo (*Voting* con *XGBost,* Regresión Logística *y Gradient Boosting*) con un pipeline. Lo siguiente es utilizar los datos adicionales que obtiene YAMNet para poder visualizar el espectrograma y la forma de la onda del audio importado.

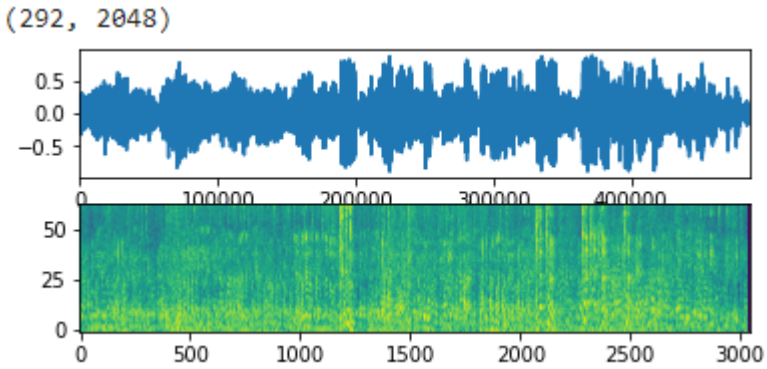


Fig. 20 Espectrograma y forma de la onda del audio importado

Para procesar el audio por partes, se utiliza este concepto de "ventana deslizante" (*rolling window*), mediante el cual va analizando todo el audio de medio segundo en medio segundo, como si fueran audios independientes. Se obtiene un dataframe con las características de todos los tramos procesados. Estos datos, se pasan al clasificador previamente entrenado el conjunto ESC-50 y utilizando la función "`predict_proba`", permite obtener la predicción si es: "tos" o "no tos" y la probabilidad en cada medio segundo. Se imprime una gráfica con las bibliotecas de Python *NumPy* y *Matplotlib* para visualizar los resultados.

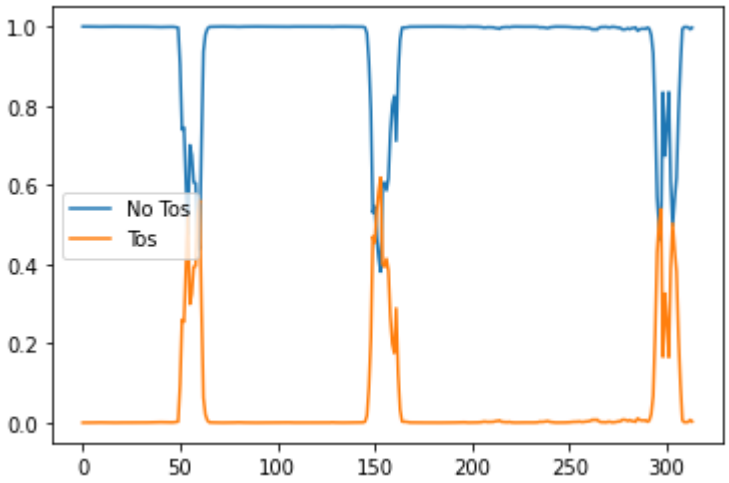


Fig. 21 Gráfica que muestra las toses detectadas por la demo del audio procesado

Con la biblioteca de *IPython* y la función *Audio*, se imprime en pantalla un reproductor que permite reproducir el audio que se ha procesado.

Fig. 22 Reproductor del audio procesado

5.5. Corolario

Los resultados que se obtuvieron con el clasificador con *Voting*, fueron muy buenos para tratarse de un conjunto desequilibrado con los 2000 audios. Esto ha permitido que la demo, sea capaz de detectar una tos con bastante precisión.

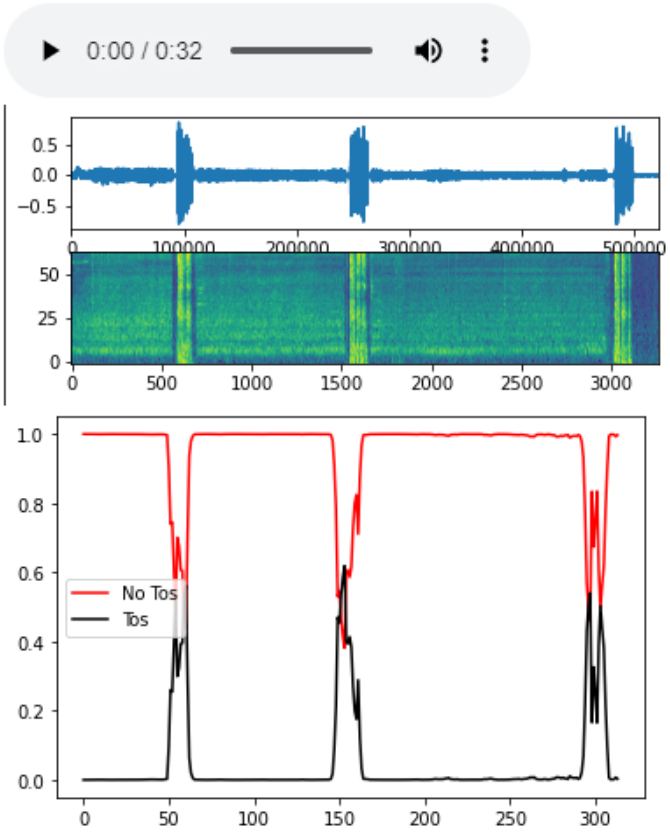


Fig. 23 Toda la información que se obtiene del audio procesado con la demo

Incluso en pruebas en las que se han introducido toses mezcladas con distintos ruidos, ha sido capaz de diferenciar muy bien las toses de otros ruidos como un timbre, una puerta, un secador e incluso con ruidos de una clase llena de gente hablando, o la calle de una ciudad ajetreada llena de coches y personas.

6. Conclusiones

La aplicación final realizada permite de manera sencilla la diferenciación de un audio de una tos de otros audios diferentes, pero tanto ella como los códigos pueden emplearse para resolver problemas de diversa índole. Pueden incluso servir de orientación para proyectos en los que no haya una clase minoritaria o en los que tan siquiera se clasifiquen audios.

Por ello confiamos que nuestro trabajo pueda ser la base de futuros proyectos más avanzados.

7. Agradecimientos

Nos gustaría agradecer a José Francisco, nuestro tutor de la universidad, toda su dedicación y apoyo ya que sin él nada de esto hubiera sido posible. También queremos mostrarnos agradecidos con Carlos López Valcárcel, quien además de habernos impartido la asignatura de matemáticas en el IES Comuneros de Castilla ha sido nuestro tutor del centro en el proyecto y nos ha proporcionado diferentes materiales con los que poder afrontar este reto.

Además, queremos agradecer a Carlos Muro todas las horas en las que nos ha permitido avanzar con los proyectos y todos sus consejos ya que sin ellos probablemente hubiéramos tardado muchísimo más en alcanzar nuestro objetivo.

Por último, nos gustaría añadir que, al estar en el BIE de Tecnologías, hemos tenido más tiempo para prepararlo con varios profesores especializados del instituto y hemos contamos con ordenadores y en general todo el equipo necesario, lo que nos ha ayudado mucho. Por ello nos gustaría dar las gracias al centro educativo IES Comuneros de Castilla.

8. Bibliografía

[1] Minería de datos. Wikipedia. La enciclopedia libre. Fecha de consulta: 20 de diciembre 2020 desde https://es.wikipedia.org/w/index.php?title=Miner%C3%ADa_de_datos&oldid=123074000

[2] *Machine Learning*. Wikipedia. La enciclopedia libre. Fecha de consulta: 20 de diciembre 2020 desde https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

[3] Clasificador (matemáticas). Wikipedia, La enciclopedia libre. Fecha de consulta: 19 de diciembre 2020 desde https://es.wikipedia.org/w/index.php?title=Clasificador_(matem%C3%A1ticas)&oldid=120668

[4] Conjunto de datos (*dataset*) Wikipedia, La enciclopedia libre. Fecha de consulta: 20 de diciembre 2020 desde https://es.wikipedia.org/wiki/Conjunto_de_datos#:~:text=Un%20conjunto%20de%20datos%20

[5] Conjunto desequilibrado sitiobigdata. Fecha de consulta: 20 de diciembre 2020 desde https://sitiobigdata.com/2019/12/24/dataset-desequilibrado-en-aprendizaje-automatico/#

[6] *Undersampling, Oversamplig* Wikipedia, La enciclopedia libre. Fecha de consulta: 21 de diciembre 2020 desde https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis

[7] *SMOTE* machinelearningmastery. Fecha de consulta: 21 de diciembre 2020 desde https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

[8] *Data Mining, the textbook.* Charu C. Aggarwal, adaptación.

[9] SMOTE relopezbriega, Matemáticas, análisis de datos y python. Fecha de consulta: 21 de diciembre 2020 desde https://relopezbriega.github.io/blog/2017/06/10/boosting-en-machine-learning-con-python/

[10] Delgado, Ronald. (2018). Introducción a la Validación Cruzada (k-fold Cross Validation) en R.

[11] VotingClassifier relopezbriega, Matemáticas, análisis de datos y python. Fecha de consulta: 28 de diciembre 2020 desde: https://medium.com/@sanchitamangale12/voting-classifier-1be10db6d7a5

[12] Python Data Analytics, Fabio Nelli, adaptación.

[13] Repositorio ESC-50 en GitHub https://github.com/karolpiczak/ESC-50

[14] VGGish GitHub. Fecha de consulta: 28 de diciembre 2020 desde: https://github.com/tensorflow/models/tree/master/research/audioset/vggish

[15] YAMNet GitHub. Fecha de consulta: 28 de diciembre 2020 desde:

https://github.com/tensorflow/models/tree/master/research/audioset/yamnet

[16] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), 2825-2830. Adaptación.

[17] Numpy Wikipedia. La enciclopedia libre. Fecha de consulta: 20 de diciembre 2020 desde :

https://es.wikipedia.org/wiki/NumPy

[18] Imbalance pypi. Fecha de consulta 3 de enero 2021 desde:

https://pypi.org/project/imbalanced-learn/

[19] Imbalance pypi. Fecha de consulta 3 de enero 2021 desde:

https://pypi.org/project/imbalanced-learn/

[20] TensorFlow Wikipedia. La enciclopedia libre. Fecha de consulta 3 de enero 2021 desde:

https://es.wikipedia.org/wiki/TensorFlow

Tabla de Ilustraciones

9. Anexo (Código)

9.1. Procesamiento del conjunto ESC-50

```
"""
  Se importan las librerías necesarias, el conjunto de audios
  ESC-50 y los archivos para la red neuronal de YAMNet
"""
!git clone https://github.com/tensorflow/models.git
!cd models/research/audioset/yamnet
!curl -O https://storage.googleapis.com/audioset/yamnet.h5
!python yamnet_test.py
!git clone https://github.com/karoldvl/ESC-50.git
!cp -r models/research/audioset/yamnet/* .
import csv
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from yamnet_test import *
from features import *
import params as yamnet_params
import yamnet as yamnet_model
import pandas as pd
import os
model = hub.load('https://tfhub.dev/google/yamnet/1')

"""
  Inicialmente, se importa el dataframe del conjunto ESC50 para obteneer los
  nombres de los audios y poder procesarlos con YAMNet
"""
metadatos_df = pd.read_csv("ESC-50/meta/esc50.csv")
names = metadatos_df.filename
```

```
last = metadatos_df.fold.iloc [-1]+1

"""
  La función "yamnet_caracteristicas", procesa todos los audios del conjunto
  de forma sucesiva, haciendo la media y la varianza de cada uno, para
  obtener una lista de 2000x2048 (2000 audios y 1024 la media de todas la
  características de los 5 segundos + 1024 la varianza de todas la
  características de los 5 segundos )
"""
def yamnet_caracteristicas(names):
  lista = []

  for i in range(len(names)):
    target = names[i]
    path = "/content/ESC-50/audio/"+target
    wav_file_name = path
    sample_rate, wav_data = wavfile.read(wav_file_name, 'rb')
    waveform = wav_data / tf.int16.max
    scores, embeddings, spectrogram = model(waveform)
    mean_embeddings = np.mean(embeddings, axis=0)
    var_embeddings = np.var(embeddings, axis=0)
    c = []
    c.extend(mean_embeddings)
    c.extend(var_embeddings)
    lista.append(c)
    print(len(lista),end=" ")
    if len(lista) % 20 == 0:
      print("")

  return np.array(lista)
lista = yamnet_caracteristicas(names)

"""
  La función "dataframe" transforma la lista con las características en una
  tabla organizada de forma que se pueda dividir de forma sencilla para la
  validación cruzada y la clasificación
"""
```

```
def dataframe(datos):
  atributos = list(map(lambda x:"Att"+str(x),range(2048)))
  metadatos_df = pd.read_csv("ESC-50/meta/esc50.csv")
  metadatos_df ["Clase"] = (metadatos_df.category == "coughing").astype(int)
  Caracteristicas_df = pd.DataFrame(datos,metadatos_df.filename,atributos)
  metadatos_df = metadatos_df.set_index("filename")
  Caracteristicas_df["fold"] = metadatos_df["fold"]
  Caracteristicas_df["Clase"] = metadatos_df["Clase"]
  Caracteristicas_df.to_csv('características(yamnet).csv')
  return Caracteristicas_df
dataframe(lista)
```

9.2. Clasificador

```
"""
 Se importan las librerías necesarias, los clasificadores y el archivo .csv
 con las características de los audios del conjunto ESC-50.
"""

from google.colab import drive
drive.mount('/content/drive')
from google.colab import files
files.upload()
!git clone https://github.com/karoldvl/ESC-50.git
from imblearn.over_sampling import SMOTE, BorderlineSMOTE
from sklearn.metrics import precision_score, recall_score,
 f1_score, roc_auc_score, accuracy_score,
  classification_report,confusion_matrix
import numpy as np
import statistics
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
import pandas as pd
```

```
from xgboost import XGBClassifier
from sklearn.ensemble import VotingClassifier
from imblearn.pipeline import make_pipeline
from joblib import dump, load
Caracteristicas_df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/Proyecto BIE/características(yamnet).csv")
"""
   Se declaran las variables globales para el reasto del código y se utiliza el
   bucle para crear los cinco  conjuntos para la validación cruzada: los cinco para entrenar(X_train,y_train), y los cinco para test(X_test,y_test).
"""
atributos = list(map(lambda x:"Att"+str(x),range(2048)))
metadatos_df = pd.read_csv("ESC-50/meta/esc50.csv")
clase = Caracteristicas_df.Clase
Fold =  {}
last = metadatos_df.fold.iloc [-1]+1
#Tuplas con  Test y Train
for i in range(1,last):

  df_test = Caracteristicas_df [Caracteristicas_df.fold == i]
  df_train =Caracteristicas_df [Caracteristicas_df.fold != i]

  X_train = df_train[atributos].values
  X_test = df_test[atributos].values

  y_train = df_train.Clase.values
  y_test = df_test.Clase.values

  Fold["Test" + str(i)] = (X_test,y_test)
  Fold["Train" + str(i)] = (X_train,y_train)

"""
   Se crea el clasificador Voting, compuesto por los otros tres
   (Regresión Logística, Gradient Boosting y XGBoost). También se crea una
   función para hacer la media de los resultados que se obtengan
   en las vallidación cruzada
"""
#Clasificador
VCBOOST = VotingClassifier(estimators=[('XGBC', XGBClassifier()),
```

```
                                        ('GBC', GradientBoostingClassifier(rando
m_state=0)),
                                        ('Lr', LogisticRegression())],
                          voting='soft', weights=[2, 1, 2])
#Medias
def media(lista):
  s = 0
  for elemento in lista:
    s += elemento
  return s / float(len(lista))

#Algoritmos utilizados para equilibrar el conjunto antes de clasificarlo
sm = SMOTE(sampling_strategy=1, random_state=42)
sm2 = BorderlineSMOTE(sampling_strategy=1, random_state=42)
rus = RandomUnderSampler(sampling_strategy=0.5,random_state=42)
ros = RandomOverSampler(sampling_strategy=0.5,random_state=42)

"""
  Con la función "imbalance", se hace la validacióin cruzada con los cinco
  conjuntos de datos que previamente se han procesado con uno de los algoritmos

  Después se crea una lista con los distintos resultados de las cinco pruebas
  en las distintas medidas que se emplean(accuracies,precision,recall,f1).
  Por último, se guarda el pipline con el clasificador en trenado en la uidad
  y se imprimen las media de todos los resultados
"""
def imbalance(name,algoritmo,clasificador):
  accuracies = []
  precision=[]
  recall=[]
  f1=[]
  pipeline = make_pipeline(algoritmo,
                           clasificador)
  for i in range(1,last):
    X_train = Fold['Train'+str(i)][0]
    y_train = Fold['Train'+str(i)][1]

    X_test = Fold['Test'+str(i)][0]
    y_test = Fold['Test'+str(i)][1]
```

```
    pipeline.fit(X_train, y_train)
    y_predict = pipeline.predict(X_test)
    #Resultados
    accuracies.append(accuracy_score(y_test, y_predict))
    precision.append(precision_score(y_test, y_predict))
    recall.append(recall_score(y_test, y_predict))
    f1.append(f1_score(y_test, y_predict))
  dump(pipeline, 'Voting.joblib')
  print()
  print(name)
  print("accuracy: {}".format(media(accuracies)))
  print("precision: {}".format(media(precision)))
  print("recall: {}".format(media(recall)))
  print("f1: {}".format(media(f1)))
  print(confusion_matrix(y_test,y_predict))
imbalance("BorderlineSMOTE",sm2,VCBOOST)
```

9.3. Demo

```
"""
  Se importan las librerías necesarias para el manejo de datos
  e imprimirlos en pantalla y archivos para la red neuronal de YAMNet.
  También se importa de GoogleDrive el pipline que contiene
  el clasificado entrenado con el conjunto ESC-50.
"""
from google.colab import drive
drive.mount('/content/drive')
!git clone https://github.com/tensorflow/models.git
!cd models/research/audioset/yamnet
!curl -O https://storage.googleapis.com/audioset/yamnet.h5
!python yamnet_test.py
!cp -r models/research/audioset/yamnet/* .
!pip install soundfile
```

```
from features import *
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
import params as yamnet_params
import yamnet as yamnet_model
import tensorflow as tf
from IPython.display import Audio
import pandas as pd
from imblearn.pipeline import make_pipeline
from joblib import dump, load
pipeline_entrenado = load('/content/drive/MyDrive/Voting.joblib')
# Se importa el archivo .wav de audio que se quiere procesar
from google.colab import files
files.upload()

"""
  La función "caracteristicas_yamnet" obtiene las características
  del audio que se importada, los datas para imprimir el espectrograma
  "spectrogram" y el "waveform" para imprimir la forma de la onda.
"""

def caracteristicas_yamnet(nombre):
  wav_file_name = nombre
  wav_data, sr = sf.read(wav_file_name, dtype=np.int16)
  waveform = wav_data / 32768.0
  params = yamnet_params.Params(sample_rate=sr, patch_hop_seconds=0.1)
  class_names = yamnet_model.class_names('yamnet_class_map.csv')
  yamnet = yamnet_model.yamnet_frames_model(params)
  yamnet.load_weights('yamnet.h5')
  scores, embeddings, spectrogram = yamnet(waveform)
  Caracteristicas_df = pd.DataFrame(np.asmatrix(embeddings))
  return Caracteristicas_df,waveform,spectrogram,wav_data,sr
Caracteristicas_df, waveform,spectrogram,wav_data,sr = caracteristicas_yamnet('
test4.wav')

"""
  Con la función "graficas", se imprime en pantalla tanto el espectrograma
```

```
  como la forma de la onda con la librería MatPlot y los datos obtenidos
  de la anterior función
"""

def graficas(formaonda,espectrograma):
  # waveform
  plt.subplot(3, 1, 1)
  plt.plot(formaonda)
  plt.xlim([0, len(formaonda)])
  # spectrogram
  espectrograma = espectrograma.numpy()
  plt.subplot(3, 1, 2)
  plt.imshow(espectrograma.T, aspect='auto',
             interpolation='nearest', origin='bottom')
graficas(waveform,spectrogram)

"""
 Con la función "identificador", se aplica el concepto de "ventana deslizanta"
 con la función rolling tanto para la media como la varianza de las
 características de medio segundo en medio segundo. (5) ya que YAMNet
 obtenía 10 cadenas de características cada segundo. A continuación,
 con el pipline guardado, se clasifica en "tos" y "no tos" cada tramo de medio
 segundo. Por último, se crea el dataframe con los resultados y se imprimen
 en pantalla y con la función "Audio" de IPython, se puede visualizar un
 reproductor par escuchar el audio
"""

def identificador(caracteristicas,clasificador):
  X_rolling = pd.concat([caracteristicas.rolling(5).var(),
                         caracteristicas.rolling(5).mean()],
                         axis=1).dropna().values
  df_predicciones = pd.DataFrame(data=clasificador.predict_proba(X_rolling))
  df_predicciones.columns = ["No Tos","Tos",]
  df_predicciones.plot(colormap='flag')
identificador(Caracteristicas_df,pipeline_entrenado)
Audio(wav_data, rate=sr)
```