Facultad
de
Ciencias

Listen To The Stadium

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Hugo Martínez Bezanilla

Director: Inés González Rodríguez

Co-Director: Adrián Fernández Roldán

Julio - 2025

RESUMEN

El análisis del sentimiento en eventos deportivos a partir del sonido ambiental permitiría comprender la relación entre eventos y la experiencia de los aficionados, lo que daría paso a mejorar la experiencia del espectador. En este proyecto se analiza la relación entre los eventos de un partido de fútbol y el sonido ambiente de las gradas, evaluando el potencial de esta conexión para el análisis emocional.

Para ello, se desarrollaron una serie de programas que estructuran los datos de los partidos a partir de archivos JSON; preparan el audio eliminando el comentarista, limpiándolo y normalizándolo para el posterior análisis;y además, ayudan a visualizar los momentos clave del partido mediante graficas relacionadas con el audio. Por otro lado, se implementó un método que detecta los posibles goles y crea una base de datos de muestra, para el posterior entrenamiento de algoritmos de machine learning para la identificación automática de goles.

Como resultado, se logra identificar los goles de los distintos equipos únicamente en posesión del sonido ambiente de un partido, demostrando así la conexión que existe entre los eventos y el sonido ambiente, sugiriendo que es posible el análisis del sentimiento a partir del audio de un acontecimiento deportivo. Por último, se muestran las dificultades encontradas y se propone un camino a seguir para la investigación como análisis del sentimiento o cómo se podría extender este análisis a otros deportes.

Palabras clave: análisis de sentimiento, sonido ambiental, fútbol, machine learning, detección de eventos.

ABSTRACT

Sentiment analysis in sports events based on ambient sound would allow us to understand the relationship between events and the fan experience, leading to improvements in the spectator's experience. This project analyzes the relationship between the events of a football match and the ambient sound from the stands to determine the extent of this connection.

To achieve this, a series of programs were developed to structure match data from JSON files, process the audio by removing the commentator, cleaning, and normalizing it for further analysis; additionally, they help visualize key moments of the match through graphs related to the audio. A method was also implemented to detect potential goals and create a sample database for the subsequent training of machine learning algorithms for automatic goal identification.

As a result, it is possible to identify goals from different teams using only the ambient sound of a match, demonstrating the existing connection between events and ambient sound. This suggests that sentiment analysis is feasible based on the audio of a sports event. Finally, the challenges encountered are presented and a path for future research is proposed like sentiment analysis or how this analysis could be extended to other sports.

Keywords: sentiment analysis, ambient sound, soccer, machine learning, event detection.

Índice general

1 Introducción

1.1. Contexto y motivación

En la actualidad, la integración de nuevas tecnologías, como puede ser el aprendizaje automático u otras técnicas de inteligencia artificial, en ámbitos de la vida cotidiana son un tema de gran relevancia. Estas tecnologías han sido ampliamente aceptadas y se han puesto de moda entre la mayoría de la población. Por ello, la integración de estas tecnologías en uno de los ámbitos más disfrutados por el público, los deportes con espectadores, centrándonos en este caso en el fútbol.

El análisis de sentimiento tiene como objetivo detectar actitudes y opiniones hacia una entidad. Aunque inicialmente se aplicaba sobre texto, hoy en día se amplía a múltiples canales, como el audio o el vídeo. Esto ha impulsado enfoques multimodales que permiten una detección más precisa de emociones gracias al uso de modelos avanzados como los basados en transformers [1]. Esta evolución está estrechamente vinculada al aprendizaje automático, ya que requiere entrenar algoritmos capaces de interpretar señales complejas y extraer patrones útiles para comprender el estado emocional de los usuarios en contextos variados.

Hoy en día existen una gran cantidad de datos que son generados durante un partido de fútbol, estos datos dinámicos han ido evolucionando con el paso del tiempo, como puede ser la posición del balón a lo largo del partido. Sin embargo, estos datos se limitan únicamente a las estadísticas tracicionales enfocadas unicamente a lo que ocurre dentro del campo, dejando completamente de lado lo que ocurre en las gradas. El entorno sonoro, compuesto por los gritos de los aficionados, los cánticos, los golpes, las reacciones y el murmullo, es una fuente inexplorada de datos que puede mostrar información valiosa sobre lo que ocurre en las gradas.

El aumento de la demanda de experiencias personalizadas, el impacto de las redes sociales y la evolución de las estrategias de marketing en el deporte hacen que atender a aspectos como el análisis de sentimiento sea cada vez más relevante. Por ejemplo, este tipo de análisis del sonido ambiente podría permitir optimizar campañas publicitarias, saber qué momentos del partido utilizar para publicaciones en redes sociales o incluso para ayudar al entrenador a saber cuando su afición está más contenta con su estilo de juego. Además, desde una perspectiva académica, la capacidad de procesar y extraer patrones significativos de señales de audio constituye una gran oportunidad para validar técnicas de aprendizaje automático en ámbitos como este.

Por todo ello, este proyecto no solo busca relacionar eventos y sentimientos con el sonido ambiental de un estadio, sino que también pretende establecer bases para un futuro desarrollo de mejora del proyecto y para posibles investigaciones que involucren datos sensoriales.

1.2. Objetivos

Actualmente, los análisis de los eventos deportivos se centran mayormente en la recolección de datos numéricos y estadísticas tradicionales, sin prestar especial atención al sonido ambiental. El reto aparece al querer desarrollar métodos que permitan limpiar y procesar correctamente el

audio, y en extraer patrones significativos que permitan diferenciar entre los distintos eventos del juego y que indiquen la respuesta emocional de los espectadores ante dichos eventos.

Objetivo general:

Desarrollar un sistema integral que, a partir del sonido ambiental captado en partidos de fútbol, permita identificar eventos relevantes (en concreto, goles) y analizar el sentimiento del público durante dichos momentos.

Objetivos específicos:

- **Detección automática de eventos:** Emplear técnicas de procesamiento de audio y aprendizaje automático para reconocer, exclusivamente a partir del sonido ambiental, cuándo se produce un gol en un partido de fútbol.
- **Análisis de sentimiento:** Realizar un estudio del estado anímico y las reacciones emocionales de los espectadores durante los goles, apoyándose en metodologías de análisis de sentimiento aplicadas al audio recogido.
- **Evaluación y validación:** Medir la precisión y fiabilidad del enfoque propuesto, identificando los principales retos y limitaciones, con el fin de sentar las bases para extender la aplicación a otros deportes y tipos de eventos deportivos en futuras líneas de investigación.

1.3. Estructura de la memoria

Esta memoria se organiza en varios capítulos que describen el desarrollo del proyecto:

- **Capítulo 1: Introducción.**
  Se expone el contexto y la motivación del proyecto, se plantea el problema, se detallan los objetivos y se presenta la estructura general de la memoria.
- **Capítulo 2: Estado del arte**
  Se realiza una revisión bibliográfica sobre técnicas de análisis de audio, procesamiento de señales y *aprendizaje* aplicados al deporte.
- **Capítulo 3: Metodología y cronología del desarrollo**
  Se describe el enfoque de la metodología adoptada, así como la cronología del desarrollo del trabajo realizado durante los meses que ha llevado el proyecto.
- **Capítulo 4: Diseño y desarrollo.**
  Se describe la arquitectura del sistema, los diferentes módulos desarrollados (preparación de datos, procesamiento del audio, visualización, detección de eventos y aplicación de algoritmos de *aprendizaje*) y las tecnologías empleadas.
- **Capítulo 5: Implementación de los modelos.**
  Se detalla el proceso de implementación de cada módulo, incluyendo aspectos técnicos, decisiones de diseño y ejemplos prácticos de integración.
- **Capítulo 6: Análisis de resultados.**
  Se presentan y discuten los resultados obtenidos en las distintas fases del proyecto, destacando la correlación entre el sonido ambiental y los eventos del partido, y se evalúa el desempeño del algoritmo de *aprendizaje*.
- **Capítulo 7: Líneas de trabajo Futuras.**
  Se describen las posibilidades de continuación del sistema desarrollado.

- **Capítulo 8: Conclusión.**
  Se sintetizan los hallazgos del proyecto y se presentan las conclusiones finales.

2 Estado del arte

2.1. Aprendizaje automático para el análisis de audios

En la actualidad, el análisis de audio se ha visto impulsado por el uso de técnicas de aprendizaje automático, las cuales han permitido automatizar procesos complejos como la clasificación de sonidos o la segmentación de fuentes acústicas. Estas técnicas, inicialmente aplicadas en campos como el reconocimiento de voz o la música, se podrían adaptar a entornos más complejos y desestructurados, como lo son los sonidos ambientales de eventos deportivos.

El audio ambiental recogido en un estadio presenta una gran riqueza de información, por ello la necesidad de encontrar la manera más eficiente de aprovechar esta información. Entre los enfoques más representativos se encuentran los métodos basados en árboles de decisión, como Random Forest y algoritmos de Boosting, conocidos por su efectividad en tareas de clasificación. Por otro lado, las redes neuronales profundas, como las Redes Neuronales Convolucionales (CNN) y las redes de memoria a corto-largo plazo (LSTM), son claves para capturar estructuras temporales y espaciales en el audio.

A continuación se exploran cada uno de estos métodos y su aplicación al análisis de audio.

2.1.1. Random forest

El algoritmo Random Forest es una técnica de aprendizaje automático supervisado basada en el concepto de ensamblado de modelos. Fue propuesto por Leo Breiman en 2001 [2] y se fundamenta en la creación de un conjunto de árboles de decisión independientes, cuya predicción conjunta mejora el rendimiento y la generalización del modelo.

A diferencia de un árbol de decisión individual, que puede sufrir de sobreajuste, Random Forest construye múltiples árboles y combina sus salidas para obtener una predicción más robusta. Para clasificación, se toma la clase más votada entre los árboles (voting), y para regresión, se calcula el promedio de las predicciones.

El entrenamiento de cada árbol se realiza utilizando dos técnicas clave:

- **Bootstrap aggregating (también conocido como bagging):** cada árbol se entrena con una muestra aleatoria del conjunto de datos original, obtenida con reemplazo.
- **Selección aleatoria de características:** en cada división del árbol, se escoge aleatoriamente un subconjunto de características para decidir la mejor separación. Esto introduce diversidad entre los árboles y reduce la correlación entre ellos.

Esta estrategia permite reducir significativamente el sobreajuste, uno de los principales problemas de los clasificadores basados en árboles individuales. Una de las ventajas de Random Forest es que permite analizar la importancia de las variables utilizadas en las predicciones, lo que ofrece una interpretación directa sobre qué características son más relevantes para el modelo.

Esta propiedad facilita la validación de las decisiones tomadas y permite optimizar futuras extracciones de características o diseñar nuevos esquemas de preprocesamiento del audio.

En el contexto del análisis de audio, Random Forest ha demostrado ser eficaz para tareas como la clasificación de eventos acústicos y la detección de patrones sonoros. Esto se debe en gran parte a su capacidad para manejar entradas de alta dimensionalidad como los coeficientes cepstrales en las frecuencias de Mel (MFCC) y otras características acústicas. Estas características, extraídas de la señal de audio, actúan como atributos numéricos que alimentan los árboles del random forest.

Un ejemplo destacado de la aplicación de Random Forest en el análisis de audio es el estudio *Acoustic Event Detection and Localization with Regression Forests* [3], donde se implementó un algoritmo de regresión basado en Random Forest para la detección y localización de eventos acústicos en flujos continuos de audio. Este enfoque permitió identificar con precisión eventos sonoros específicos en entornos ruidosos, demostrando la eficacia del modelo en situaciones del mundo real.

Random Forest es un algoritmo robusto ante datos ruidosos o con valores atípicos, ya que la agregación de múltiples modelos reduce la sensibilidad a anomalías individuales. Otra fortaleza destacada es su capacidad para manejar conjuntos de datos con un gran número de variables, sin requerir una fuerte selección previa de características, ya que incorpora de forma intrínseca un mecanismo de evaluación de la importancia de cada variable. Asimismo, Random Forest puede aplicarse tanto a problemas de clasificación como de regresión, y suele ofrecer un buen rendimiento sin necesidad de ajustar muchos parámetros.

2.1.2. Boosting

Boosting es una técnica de aprendizaje automático supervisado que tiene como objetivo combinar múltiples clasificadores débiles, modelos que apenas superan el rendimiento aleatorio, para formar un clasificador fuerte. A diferencia de métodos como el bagging, que entrenan modelos de forma paralela e independiente (como Random Forest), el boosting entrena sus modelos de manera secuencial, donde cada nuevo modelo intenta corregir los errores cometidos por los anteriores.

El principio fundamental del boosting consiste en asignar mayor peso a las muestras mal clasificadas por los modelos anteriores. Así, el algoritmo va centrando progresivamente su atención en los ejemplos más difíciles. Al final, las predicciones de todos los modelos se combinan mediante un esquema de votación ponderada (en clasificación) o promedio (en regresión), mostrado en la Figura 2.1.

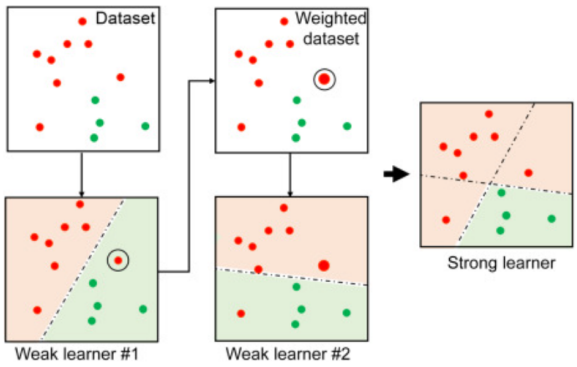

Figura 2.1: Implementación de Ada Boost [4].

Existen varias implementaciones de boosting, cada una con características particulares. El algoritmo más conocido es AdaBoost (Adaptive Boosting), que fue introducido por Freund y Schapire en 1997 [5]. AdaBoost utiliza clasificadores débiles y ajusta iterativamente los pesos de las muestras de entrenamiento, dando mayor importancia a las muestras mal clasificadas en cada paso. Gradient Boosting es otra variante, que utiliza el descenso de gradiente para minimizar una función de pérdida arbitraria, lo que le permite ser extremadamente flexible y adaptable a diferentes tipos de problemas.

En la misma línea, implementaciones como XGBoost, LightGBM y CatBoost son versiones optimizadas de Gradient Boosting, con mejoras significativas en velocidad, manejo de datos faltantes y regularización.

Una de las principales ventajas del boosting es su alta precisión predictiva, especialmente en conjuntos de datos complejos y no lineales. A través de su enfoque secuencial, el boosting es capaz de aprender patrones complejos en los datos que otros algoritmos podrían pasar por alto. Además, el boosting es particularmente útil cuando se enfrenta a problemas de clases desbalanceadas o cuando los datos contienen ruido, ya que el algoritmo puede enfocar su aprendizaje en los errores más difíciles y relevantes.

Otra ventaja es su flexibilidad para adaptarse a diferentes tipos de datos y tareas, ya que permite modificar la función de pérdida y la técnica de optimización utilizada. Finalmente, el boosting tiene la capacidad de ofrecer modelos altamente interpretables al analizar los pesos asignados a las diferentes características, lo que permite una mejor comprensión de cómo el modelo toma decisiones.

2.1.3. Long Short-Term Memory (LSTM)

Las redes neuronales recurrentes (*Recurrent Neural Networks*, RNN) son una clase de redes neuronales diseñadas específicamente para procesar datos secuenciales, como series temporales, texto o audio. Las RNN tienen conexiones recurrentes que permiten que la información fluya de una iteración temporal a otra, lo que les otorga una "memoria" sobre pasos anteriores en la secuencia. Esto las hace especialmente útiles para tareas en las que el contexto y el orden de los datos son fundamentales

Las Long Short-Term Memory networks (LSTM) son un tipo de red neuronal recurrente (RNN) diseñada para procesar y modelar datos secuenciales. Su principal ventaja es su capacidad para mantener información a largo plazo, algo que las RNN tradicionales no pueden hacer de manera efectiva debido a los problemas de *vanishing gradient*[6] o el *exploding gradient*[7]. Estos problemas

dificultan el entrenamiento en secuencias largas, ya que los gradientes se vuelven o demasiado pequeños o demasiado grandes, impidiendo que la red aprenda relaciones a largo plazo entre elementos de la secuencia.

Las LSTM, introducidas por Hochreiter y Schmidhuber en 1997 [6], abordan este problema mediante una arquitectura de celdas de memoria que mantienen su estado a lo largo del tiempo y que están controladas por tres compuertas: la compuerta de entrada (input gate), la compuerta de olvido (forget gate) y la compuerta de salida (output gate). Estas compuertas actúan como mecanismos de control que determinan cuándo se debe agregar, mantener o eliminar información del estado interno de la célula, utilizando funciones de activación (sigmoide y tangente hiperbólica) para decidir qué parte de la información pasa o se bloquea.

- **Forget gate** decide qué información del estado anterior debe eliminarse, permitiendo a la red olvidar datos que ya no son relevantes.
- **Input gate** regula qué nueva información se almacena en la celda.
- **Output gate** etermina qué parte del estado de la celda debe usarse como salida y como entrada para la siguiente capa o unidad LSTM.

Un bloque de memoria estándar en una red LSTM contiene al menos una célula con una conexión recurrente a sí misma, conocida como *Constant Error Carousel* (CEC), la cual tiene un peso fijo de valor 1. Esta conexión permite que el estado interno de la célula, denotado como s_c, se mantenga a lo largo del tiempo, lo que resulta clave para preservar información a largo plazo. El acceso de lectura y escritura a esta célula está regulado por dos compuertas: la compuerta de entrada (*input gate*, y_{in}) y la compuerta de salida (*output gate*, y_{out}).

La actualización del estado de la célula se realiza combinando el valor de entrada —transformado mediante una función de activación, representado como g— con la activación de la compuerta de entrada y_{in}, y sumando dicho producto al estado anterior de la célula, $s_c(t-1)$. Finalmente, la salida de la célula se calcula multiplicando el estado actualizado s_c por la activación de la compuerta de salida y_{out}, lo que permite controlar cuánta información del estado interno se transmite al siguiente nivel de procesamiento. Una representación visual del bloque de memoria se muestra en la Figura 6.3 .

[6] Situación en la que los gradientes se hacen demasiado pequeños durante el entrenamiento, impidiendo que la red aprenda correctamente relaciones a largo plazo.

[7] Ocurre cuando los gradientes crecen de forma excesiva, provocando inestabilidad en el entrenamiento del modelo.

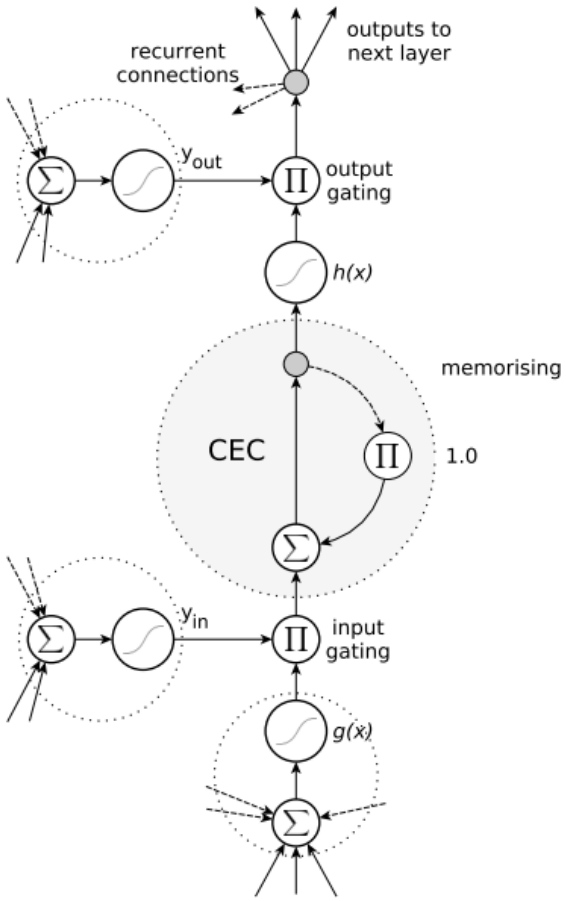

Figura 2.2: Bloque de memoria LSTM estandar. [7].

Este diseño permite a las LSTM modelar dependencias de largo plazo en secuencias de datos, lo que las hace especialmente útiles para tareas como procesamiento de lenguaje natural (NLP), predicción de series temporales y, en el contexto de tu trabajo, análisis de audio.En aplicaciones de audio, las LSTM son especialmente valiosas para capturar la dinámica temporal de una señal acústica, analizando cómo evoluciona el sonido en el tiempo. Generalmente, estas redes trabajan sobre características acústicas extraídas previamente, como los *Mel-frequency cepstral coefficients (MFCCs)*, características extraídas del audio que representan la envolvente espectral de una señal, utilizando una escala de frecuencias mel, comúnmente empleadas en reconocimiento de voz y procesamiento de audio.

2.1.4. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales (CNN) son un tipo de red neuronal profunda especialmente diseñadas para el procesamiento de datos con una estructura de grid, como las imágenes. Su principal característica es el uso de una operación matemática llamada convolución, que permite detectar patrones espaciales, como bordes, texturas o formas, dentro de una imagen. Esta operación es aplicada a través de filtros (o kernels), que recorren la imagen de manera local, generando mapas de características que capturan información relevante.

Una CNN típicamente está compuesta por varias capas:

- **Capa convolucional (Convolutional Layer):** aplica los filtros para extraer características espaciales de la imagen de entrada. Cada filtro detecta un patrón particular, y los

resultados se combinan para formar un mapa de activación.

- **Capa de activación (Activation Layer):** generalmente se utiliza la función *ReLU (Rectified Linear Unit)*[9] para introducir no linealidad en el modelo y permitirle aprender patrones más complejos.
- **Capa de agrupamiento (Pooling Layer):** reduce las dimensiones de los mapas de activación mediante operaciones de max pooling o average pooling, lo que disminuye la carga computacional y ayuda a evitar el sobreajuste.
- **Capa completamente conectada (Fully Connected Layer):** después de varias capas convolucionales y de pooling, la red conecta todas las activaciones a una capa densa, lo que permite realizar tareas como clasificación o regresión.

Las CNN son ampliamente utilizadas en tareas de reconocimiento de imágenes y procesamiento de señales debido a su capacidad para aprender representaciones jerárquicas de datos. En el caso del análisis de audio, las CNN también se emplean para extraer características de espectrogramas, transformando señales de audio en imágenes donde las CNN pueden identificar patrones relacionados con sonidos específicos. En la Figura 2.3 se muestra visualmente como se realiza la operación de convolución.

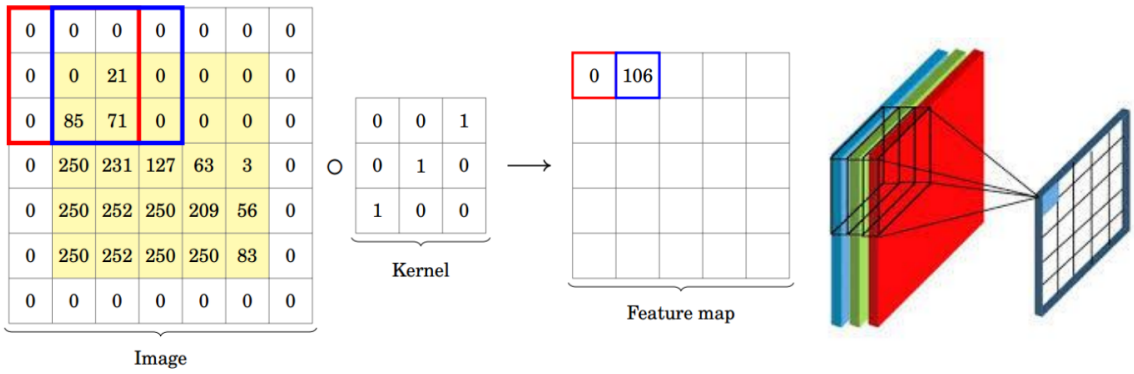

Figura 2.3: Operación de convolución. [8].

Estas redes neuronales presentan ciertas ventajas en el procesamiento de datos en forma de imágenes o secuencias espaciales. Una de sus principales ventajas es la capacidad de detectar automáticamente características relevantes de los datos, como bordes, texturas o patrones complejos. Además, las CNN son especialmente eficaces para capturar jerarquías espaciales, lo que les permite aprender representaciones de alto nivel. Otra ventaja importante es la compartición de pesos, que reduce significativamente el número de parámetros a entrenar, haciendo que las CNN sean más eficientes en términos computacionales.

2.1.5. Transformadores

Los transformadores son arquitecturas basadas en mecanismos de atención que procesan secuencias completas en paralelo y capturan relaciones de largo alcance entre elementos.

[9]ReLU (Rectified Linear Unit): es una función de activación que establece a cero cualquier valor negativo de la entrada y mantiene los valores positivos sin cambios, lo que introduce no linealidad en el modelo y ayuda a acelerar el entrenamiento.

Auto-atención La auto-atención calcula pesos de relevancia entre cada par de posiciones de la misma secuencia, integrando información de todo el contexto. Esto permite que cada elemento de la secuencia vea y ajuste su representación según otros elementos relevantes, mejorando la captura de dependencias a cualquier distancia.

Atención cruzada La atención cruzada conecta dos secuencias diferentes: una actúa como consultas (queries) y la otra como claves/valores (keys/values). En aplicaciones multimodales, por ejemplo, se puede usar un espectrograma para generar queries y la señal de audio en dominio temporal como keys/values, fusionando información de ambas fuentes.

Dominio cruzado El enfoque de dominio cruzado aprovecha lo aprendido en un espacio (p. ej., frecuencia) para enriquecer otro (p. ej., tiempo). Al combinar representaciones de distintos dominios, el modelo generaliza mejor y capta patrones que serían difíciles de descubrir individualmente.

Cada capa de transformer combina:

- Múltiples cabezas de atención (auto-atención o atención cruzada) en paralelo.
- Conexiones residuales y normalización de capa para estabilizar y acelerar el entrenamiento.
- Una red feed-forward aplicada de forma independiente a cada posición para aumentar la complejidad expresiva.

2.1.6. Métricas de evaluación del modelo en la clasificación de goles

1. **Exactitud (Accuracy):**
   La exactitud se define como la proporción de predicciones correctas (tanto verdaderos positivos como verdaderos negativos) respecto al total de muestras evaluadas. Aunque resulta intuitiva y ofrece una visión global del rendimiento, su interpretación puede ser engañosa cuando existe un sesgo en la distribución de clases (por ejemplo, si predominan los segmentos sin gol frente a los segmentos con gol).

2. **Precisión (Precision):**
   La precisión mide la proporción de casos clasificados como "gol" que realmente corresponden a un gol auténtico. Matemáticamente, se expresa como:

   $$\text{Precision} = \frac{\text{VP}}{\text{VP} + \text{FP}},$$

   donde VP (verdaderos positivos) son los goles correctamente detectados y FP (falsos positivos) son los segmentos sin gol erróneamente etiquetados como evento. Una precisión elevada indica que el modelo comete pocos falsos positivos.

3. **Sensibilidad (Recall):**
   El recall cuantifica la proporción de goles reales que el modelo logra identificar. Su fórmula es:

   $$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}},$$

   donde FN (falsos negativos) son los goles que el modelo no llegó a detectar.

4. **F1-Score:**
   El F1-score es la media armónica entre precisión y recall, calculada mediante:

   $$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Dado que penaliza tanto los falsos positivos como los falsos negativos, resulta especialmente útil cuando existe un desequilibrio entre clases (por ejemplo, cuando los segmentos de "sin gol" superan ampliamente a los de "gol"). Un alto valor de F1-score indica un balance adecuado entre ambas dimensiones.

2.2. Análisis de sentimientos

El análisis de sentimientos, también conocido como minería de opiniones, es una rama del procesamiento del lenguaje natural (NLP) que se enfoca en identificar, extraer y clasificar las emociones expresadas en un fragmento de información, comúnmente texto. Su objetivo principal es determinar la actitud de un hablante o escritor con respecto a un tema específico, la polaridad (positiva, negativa, neutral) y, en ocasiones, la intensidad del sentimiento.

A medida que el análisis de sentimientos ha evolucionado, también ha crecido su aplicación en medios no textuales, como imágenes, vídeo y, especialmente, audio, lo cual introduce nuevos desafíos y oportunidades en el estudio de señales.

Cuando el análisis de sentimientos se aplica al audio, el proceso se centra en la interpretación emocional de las características acústicas presentes en una señal sonora. Este enfoque, también conocido como análisis de sentimientos paralingüístico, intenta inferir el estado emocional de una persona, o en el caso de este trabajo, de una multitud, a partir de señales no verbales, como la entonación, la intensidad, el ritmo o el timbre.

Como se describe en los trabajos de Das y Singh (2023)[1], así como en la revisión de Mehta (2021)[9], el análisis de audio permite captar aspectos paralingüísticos como el tono, la entonación, la velocidad del habla o incluso elementos no verbales como risas o aplausos, los cuales son fundamentales para detectar emociones. Aunque muchas de las investigaciones en este campo se han centrado en el análisis de voz, especialmente en contextos como entrevistas o llamadas telefónicas, también se ha explorado el potencial del audio ambiental, un área en crecimiento que ofrece posibilidades particularmente relevantes para eventos deportivos.

En este contexto, el sonido ambiente de un estadio puede actuar como un indicador emocional colectivo de gran valor. Así, el análisis de sentimientos aplicado al audio ambiental permite inferir el estado emocional general de una afición durante un partido, sin necesidad de procesar lenguaje verbal directamente.
El análisis de sentimientos es un proceso complejo que involucra varias etapas clave, cuyo objetivo es identificar, extraer y clasificar emociones latentes en los datos. En el caso concreto del audio , los pasos se desarrollan de la siguiente forma:

- **Extracción de características acústicas:** Una vez procesado el audio, se extraen características que pueden indicar información emocional. Esto incluye tanto atributos de bajo nivel como el tono, energía, tempo o espectrogramas, como características de alto nivel que modelan patrones temporales y frecuenciales.
- **Representación temporal y contextual:** El audio es una señal continua y altamente dependiente del tiempo. Por tanto, resulta clave representar la señal teniendo en cuenta su evolución temporal. Esto se puede lograr mediante ventanas móviles o secuencias de frames. La relación contextual entre momentos de la grabación, por ejemplo, el aumento repentino de volumen seguido de un canto sostenido, puede indicar momentos clave del evento como un gol o una acción polémica.
- **Clasificación del sentimiento:** En esta fase, se utilizan modelos de machine learning o deep learning para etiquetar las emociones predominantes. En lugar de centrarse en

emociones individuales, en el caso del audio ambiental puede ser más útil un enfoque de nivel de emoción.

A diferencia del análisis de voz individual, el procesamiento del audio de multitudes en eventos deportivos requiere enfoques más robustos que puedan trabajar con señales ruidosas, superposiciones de voces y variabilidad acústica. Por ello, se han propuesto métodos basados en aprendizaje profundo multimodal que combinan el análisis acústico con otras señales (como vídeo o texto), aunque en este trabajo se prioriza únicamente la señal sonora.

2.3. Procesamiento de audio en eventos deportivos: eliminación del comentarista, limpieza y normalización

El análisis del sonido ambiental en eventos deportivos requiere de un procesamiento del audio para separar el sonido ambiente del comentarista, eliminar interferencias no deseadas y garantizar la uniformidad en los niveles de amplitud. Por lo tanto, la eliminación del comentarista, la limpieza del audio y su normalización son procesos esenciales para conseguir un audio adecuado para su posterior análisis.

2.3.1. Separación de las fuentes de sonido

La eliminación de la voz del comentarista en eventos deportivos es un paso clave en el procesamiento del audio, ya que sin ello no es posible extraer información relevante del sonido ambiente sin interferencias externas. Los avances recientes en inteligencia artificial han permitido el desarrollo de modelos basados en redes neuronales profundas que permiten una buena separación de las distintas fuentes de sonido en una señal de audio. Sin embargo, la mayoría de estos modelos se han orientado principalmente al ámbito musical.

En este contexto, aparecen dos herramientas de código abierto de una gran relevancia, como son Spleeter[1] y Demucs[2]. Spleeter, desarrollado por Deezer, utiliza *redes neuronales convolucionales* para realizar una separación de fuentes en el ámbito musical, lo que permite extraer componentes como la voz e instrumentos en grabaciones musicales.

Por otro lado, Demucs es otra herramienta basada en aprendizaje profundo que, al igual que Spleeter, tiene una gran capacidad para separar correctamente el audio en distintas pistas. Demucs, en concreto, realiza la separación en cuatro fuentes: batería, bajo, guitarra y voz. Ambas herramientas podrían servir para facilitar la extracción del sonido ambiente del estadio, separando de este la voz del comentarista.

Tras evaluar varias soluciones de código abierto para la separación de fuentes de audio (incluyendo Spleeter y otras alternativas basadas en espectrogramas), los dos modelos que ofrecieron un rendimiento superior fueron Spleeter y Demucs. No obstante, en las pruebas específicas de aislamiento de la voz del comentarista frente al sonido ambiente de un estadio, Demucs superó con creces a Spleeter, obteniendo una separación más limpia y con menos artefactos, por lo que se seleccionó como la herramienta óptima para este trabajo.

Ya que la herramienta utilizada en este trabajo es Demucs nos centramos en ella. Demucs es un modelo de separación de fuentes que se basa en una arquitectura convolucional U-Net inspirada en Wave-U-Net, tal y como se explica a continuación.

[1] https://github.com/deezer/spleeter
[2] https://github.com/facebookresearch/demucs

Wave-U-Net

Es una red neuronal convolucional centrada en tareas de separación de fuentes de audio,la cual trabaja directamente sobre la forma de onda de audio sin procesar. Wave-U-Net es una adaptación de la arquitectura U-Net al dominio temporal unidimensional para realizar una separación integral de fuentes de audio [10].

Una señal de audio puede representarse en el dominio de la frecuencia mediante dos componentes principales: el espectro de magnitud, que indica cuánta energía hay en cada frecuencia, y la información de fase, que describe cómo se combinan las distintas frecuencias a lo largo del tiempo. Los modelos para la separación de fuentes de audio normalmente operan solo sobre el espectro de magnitud, ignorando la fase y, con ello, parte de la estructura temporal de la señal. En cambio, la red Web-U-Net permite modelar también la información de fase, evitando así el uso de transformaciones espectrales fijas. Wave-U-Net remuestrea repetidamente mapas de características para calcular y combinar información en diferentes escalas de tiempo.

Los resultados del uso de este modelo en la separación de voces cantadas indican que ofrece un alto rendimiento comparable al de otras arquitecturas de vanguardia basadas en espectogramas con los mismos datos. Sin embargo, existen problemas con valores atípicos en las métricas de evaluación de SDR[2] utilizadas.

Demucs v4

Actualmente, Demucs se trata de un modelo híbrido de separación de espectograma y waveform que utiliza transformadores, donde las capas más internas se reemplazan por un transformador de *dominio cruzado*, utilizando *autoatención* (self-attention en inglés) dentro de un dominio y *atención cruzada* (cross-attention en inglés) entre dominios.

La señal de entrada en Demucs se procesa de manera paralela mediante dos vías: una temporal y otra espectral. Por un lado, la onda original se introduce en un codificador temporal que opera directamente en el dominio del tiempo. Por otro lado, la señal también se transforma mediante una Transformada Rápida de Fourier (STFT) y el espectrograma resultante se procesa a través de un codificador espectral.

Ambas representaciones, temporal y espectral, se combinan posteriormente en los puntos donde sus dimensiones coinciden, permitiendo al modelo integrar información complementaria de ambos dominios. A continuación, la arquitectura emplea un decodificador simétrico que reconstruye la señal separada. El espectrograma generado se transforma de nuevo al dominio temporal mediante la transformada inversa (ISTFT) y se suma a la salida obtenida del camino temporal, generando así la señal final procesada. Este proceso esta representado en la Figura 2.4.

[2]La evaluación SDR (Signal-to-Distortion Ratio, o Relación Señal-Distorsión) es una métrica utilizada para medir la calidad de la separación de fuentes en procesamiento de audio.

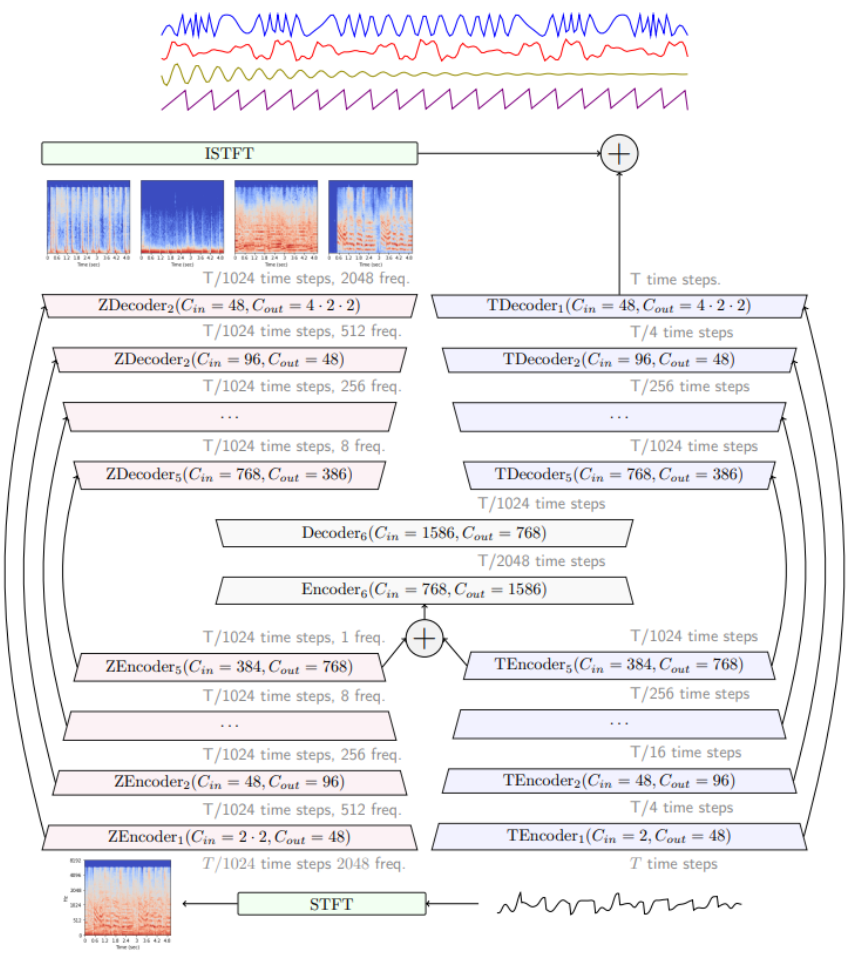

Figura 2.4: Estructura híbrida de Demucs.El prefijo Z se usa para las capas espectrales, y el prefijo T para las capas temporales. [11]

El modelo se ha entrenado utilizando grandes cantidades de datos de música multicanal para aprender a identificar las distintas fuentes. Los resultados experimentales han demostrado que Demucs es capaz de separar las fuentes de audio con un alto grado de precisión, pero sin la necesidad de realizar transformaciones espectrales fijas.

A través de su arquitectura y su capacidad de modelar tanto la fase como la magnitud del audio, Demucs puede superar algunas de las limitaciones de otros modelos que solo procesan la magnitud del espectro. La fase, que se suele ignorar, es crucial para una separación de audio de alta calidad, ya que influye directamente en la percepción del sonido original. [11]

2.3.2. Limpieza de audio y reducción de ruido

Después de eliminar el comentarista, es necesario aplicar técnicas de limpieza de audio y reducción de ruido para desacerse de sonidos inoportunos sin comprometer la calidad del sonido ambiental. Dado que el sonido que se quiere analizar está formado por el bullicio de la multitud, esta limplieza de audio debe ser suficientemente precisa como para eliminar picos no deseados de amplitud, pero lo más leve para no perder este sonido ambiental que en muchos otros contextos sería considerado ruido.

Este enfoque representa un desafío, debido a que muchas de las técnicas de reducción de ruido estan diseñadas para eliminar el ruido de fondo en contextos donde este es considerado indesea-

ble, como en grabaciones de voz. Sin embargo, en este caso, el ruido ambiental es el que resulta de interés, por lo que se busca un tratamiento del audio especializado que preserve los sonidos buscados mientras se eliminan los no deseados.

Algunos de los problemas a solventar son:

- Distorsiones propias de la grabación, producidas por el equipo de captura.
- "Pops"de micrófono, que quedan tras la eliminación del comentarista.
- Interferencias eléctricas, por mala conexión del equipo.

Para abordar estos problemas, se utilizan distintas técnicas de filtrado y procesamiento de la señal que se explican a continuación.

Filtrado de frecuencias: filtros de paso-alto y paso-bajo

El filtrado de frecuencias es una de las técnicas más básicas y efectivas para eliminar ruido en el audio. Existen dos tipos principales de filtros que permiten reducir ciertas frecuencias no deseadas.

Filtros de paso-alto

Son utilizados para eliminar frecuencias bajas indeseadas, como vibraciones del micrófono, ruidos de golpes o sonidos de baja frecuencia que se generan en el ambiente. En el caso concreto de este trabajo no resultaron de utilidad, ya que lejos de servir a la reducción de ruido, deterioraban la señal y afectaban la fidelidad del contenido útil.

Existen enfoques más avanzados, como el ajuste dinámico de la frecuencia de corte, el filtrado en el dominio tiempo-frecuencia, el filtrado por ventana deslizante con estimación de ruido y la detección de eventos con enmascaramiento espectral, que prometen optimizar la atenuación de ruidos no deseados sin comprometer la calidad del audio restante. Sin embargo, aunque fueron considerados para su aplicación en este trabajo, finalmente no llegaron a implementarse, dado que en las pruebas iniciales no mostraron una mejora significativa en la relación señal-ruido y presentaron un comportamiento similar al de los filtros de paso-alto convencionales, introduciendo igualmente artefactos indeseados. [12]

Filtros de paso-bajo

Los filtros de paso-bajo desempeñan un papel clave en la limpieza del audio, ya que permiten atenuar las frecuencias altas no deseadas, eliminando así artefactos como los pops de micrófono, interferencias eléctricas y otros ruidos de alta frecuencia que pueden afectar la calidad del sonido ambiental. En el contexto del análisis de eventos deportivos, donde el sonido de la multitud y el ambiente del estadio son esenciales, estos filtros ayudan a suavizar el espectro de frecuencias sin comprometer la información relevante.

La reducción de ruido mediante estos filtros se basa en el hecho de que muchas fuentes de interferencia acústica tienen su energía concentrada en el rango de altas frecuencias. Aplicar un filtro de paso-bajo permite preservar las frecuencias más bajas, que suelen contener los gritos del público, aplausos y otras manifestaciones sonoras características de un evento deportivo.

Para lograr una limpieza efectiva sin distorsionar el sonido original, es común emplear filtros de paso-bajo de segundo orden, los cuales ofrecen una transición más controlada entre las frecuencias

que se deben preservar y las que se desean atenuar. En pruebas aplicadas en grabaciones de audio, se ha observado que estableciendo un umbral de filtrado en frecuencias altas, se consigue una notable reducción del ruido sin afectar la percepción natural del sonido ambiente [13].

Reducción de ruido espectral

Una técnica más avanzada para la limpieza de audio es la reducción de ruido espectral, esta técnica consiste en analizar las frecuencias de la señal de audio y eliminar las componentes indeseadas sin afectar el resto de la señal. Se basa en la Transformada Rápida de Fourier (FTT) para identificar patrones de ruido y atenuarlos.

El procedimiento clásico parte del análisis espectral de la señal mediante la Transformada Rápida de Fourier, sobre la cual se construye un perfil de ruido estacionario, generalmente extraído de regiones de silencio o pausas identificadas dentro del audio. Este perfil se sustrae posteriormente en cada ventana temporal mediante técnicas de sustracción espectral, atenuando las bandas frecuenciales en las que se encuentra el ruido.

Para evitar artefactos de procesamiento como el musical noise, distorsiones generadas por una eliminación agresiva de ruido, se aplican técnicas de suavizado espectral y filtrado post-procesamiento. Además, estos métodos suelen ir acompañados de algoritmos de reconstrucción temporal para mantener la coherencia del audio procesado, evitando pérdidas bruscas de energía sonora que afectarían al análisis posterior del sonido ambiente. El procedimiento completo se ilustra en la Figura 2.5. [14]

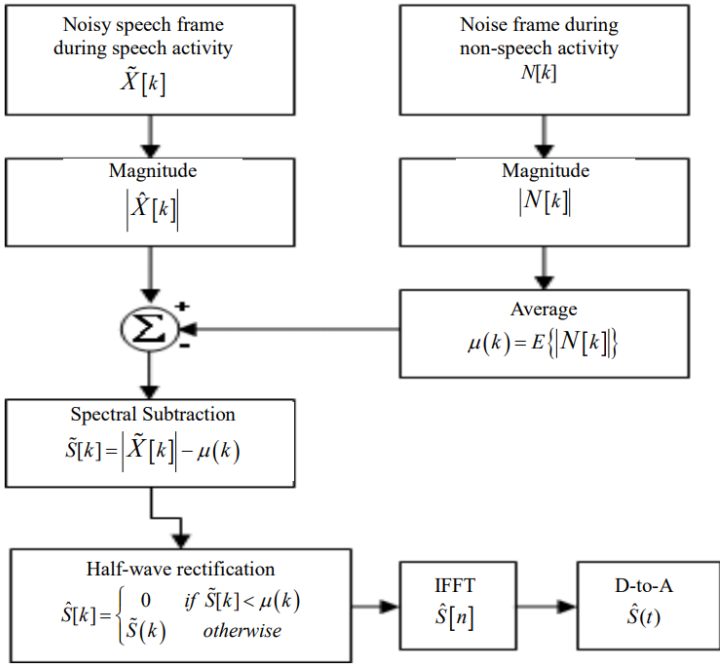

Figura 2.5: Diagrama de flujo de eliminación de ruido por sustracción espectral. [14]

2.3.3. Normalización del audio para consistencia en el análisis

La normalización del audio es un proceso fundamental para garantizar que los niveles de amplitud del sonido ambiental sean consistentes a lo largo de la grabación. Esto es especialmente importante en transmisiones deportivas, donde las variaciones de volumen pueden generar pro-

blemas en el análisis automático de eventos.

Existen diversos enfoques y herramientas para lograr esta normalización de manera eficiente. Algunos de los métodos más comunes implican ajustar los niveles de audio basándose en la potencia de la señal, de manera que el volumen se mantenga dentro de un rango determinado sin distorsionar el contenido sonoro. Por ejemplo, algunos estudios han mostrado cómo herramientas de normalización automática son capaces de regular el volumen de los archivos de audio de manera que se asegura una calidad uniforme, incluso cuando se trabaja con grabaciones de diferentes fuentes.

Este enfoque es aún más importante cuando se trabajan con grabaciones que contienen picos inesperados que pueden surgir de equipos de grabación, interferencias durante la transmisión o como es el caso de este proyecto de la eliminación del comentarista. En estos casos, la normalización no solo mejora la claridad del sonido ambiental, sino que también facilita el análisis posterior.[15]

3 Metodología y cronología del desarrollo

3.1. Enfoque de la metodología

Durante el desarrollo del proyecto ha adoptado un enfoque basado en metodologías ágiles, concretamente inspiradas en el marco de trabajo Scrum [16] y Kanban [17]. Las metodologías ágiles priorizan la entrega continua y la colaboración iterativa sobre una planificación rígida, lo que resulta especialmente útil en entornos donde los requisitos pueden evolucionar con el tiempo.

Las metodologías ágiles se fundamentan en la búsqueda de entregas frecuentes y en la capacidad de adaptarse rápidamente a los cambios. En lugar de planificar un único lanzamiento al final del proyecto, se divide el trabajo en ciclos cortos y manejables. Esto favorece a la detección temprana de errores y la incorporación de mejoras. La comunicación continua de los integrantes, garantizan que los requisitos evolucionen de acuerdo a sus observaciones, evitando desviaciones costosas o tardías. Al mismo tiempo, la metodología ágil promueve la organización y la responsabilidad de cada miembro del equipo.

Por un lado, Scrum aporta una estructura basada en ciclos de trabajo cortos (sprints), reuniones de seguimiento y revisiones periódicas para presentar incrementos funcionales del software. Esta cadencia fija de sprints facilita la planificación a corto plazo, la detección temprana de errores y la adaptación rápida de prioridades.

Por su parte, Kanban introduce un sistema de flujo continuo y visualización del estado de cada tarea a través de un tablero de columnas ("Por hacer", "En progreso", "Hecho"). Al limitar el número de tareas en curso (Work-In-Progress), Kanban ayuda a evitar cuellos de botella y a equilibrar la carga de trabajo en cada etapa.

La combinación de ambos métodos ha permitido, por un lado, respetar hitos y entregas periódicas con la disciplina de Scrum, y por otro, mantener la flexibilidad de Kanban para incorporar cualquier cambio inesperado sin esperar al cierre de un sprint.

Los elementos clave de esta metodología incluyen:

- **Sprints:** cada sprint corresponde a un ciclo de trabajo corto de duración, al final del cual se revisan los avances, se actualizan los objetivos y se planifican las tareas del siguiente sprint.
- **Milestones:** objetivos de alto nivel que marcan fases importantes del proyecto.
- **Issues:** unidades individuales de trabajo que representan tareas, errores o nuevas funcionalidades.
- **Reuniones regulares:** sesiones periódicas para evaluar el progreso, resolver bloqueos y planificar los siguientes pasos.
- **Documentación viva:** mediante el uso de wikis o repositorios colaborativos para ir mostrando los avances.

3.2. Aplicación de la metodología agil

Para la implementación práctica de esta metodología se utilizó GitLab. En esta plataforma se organizaron los sprints formales a través de milestones, cada uno de los cuales contenía un conjunto de issues que se iban marcando como "abiertos", "en progreso" o "cerrados" conforme avanzaba el trabajo. Estos estados se representaban visualmente en tableros Kanban, lo que facilitaba la visión global del flujo de tareas y su prioridad.

Además, se hizo uso de la sección Wiki de Gitlab para ir documentando los resultados, decisiones de diseño y tablas de datos, facilitando la trazabilidad de los avances y resultados obtenidos.

Se realizó un seguimiento semanal del proyecto mediante reuniones regulares en las que participábamos las tres personas que formábamos el equipo: yo, como desarrollador principal, y mis dos co-directores de HP, que asumían funciones de supervisión y orientación (equivalentes a Product Owner y Scrum Master en un entorno ágil). Estas reuniones servían para analizar los progresos, definir nuevos objetivos y añadir issues correspondientes al avance del proyecto. Este sistema ha permitido mantener una visión clara del estado del proyecto y una adaptación flexible ante los descubrimientos y retos que surgieron.

3.3. Cronología del desarrollo

El desarrollo del trabajo se fue realizando de manera iterativa y progresiva. A continuación, se presenta una cronología de los hitos alcanzados y cómo evolucionaron los objetivos a medida que avanzaba el trabajo.

- **Octubre:** El primer paso fue familiarizarse con el entorno de trabajo en GitLab, aprendiendo a utilizar herramientas como los issues, las ramas y los tableros Kanban. Paralelamente, se comenzó la exploración inicial de los datos disponibles: archivos en formato .txt los cuales contenían datos sobre los partidos, jugadores, equipos y eventos del partido, junto con archivos de audio asociados, los cuales en un principio no poseían comentarista. Durante esta fase se diseñó el primer código para leer y procesar la información básica de los encuentros.
- **Noviembre:** El foco se trasladó a la visualización de los eventos extraídos de los archivos, generando gráficos que representaban tanto su distribución temporal como su relación con los equipos y distribuciones espaciales. Por otro lado, a finales de mes el tipo de archivos que contenían los eventos pasó a ser de tipo .xml. Se abordó un problema que surgió con los caracteres especiales en los nombres y se establecieron mecanismos para mapear correctamente a los jugadores con sus respectivos clubes. Se añadieron mejoras para dividir los eventos en categorías por jugador o por equipo y se desarrollaron los primeros gráficos superpuestos sobre la onda de sonido.
- **Diciembre:** Se comenzó a experimentar con la diferenciación de tipos de cánticos en función de características tonales, con el objetivo de detectar automáticamente patrones acústicos que indicaran situaciones relevantes del partido. También se desarrollo un programa para extraer los distintos audios de los eventos diferenciandolos por equipos.
- **Enero:** La obtención de grabaciones sin la voz del comentarista pasó a ser un problema, lo que motivó el desarrollo del programa Audio Cleaner, diseñado para separar fuentes de sonido. Además, se produjo un cambio en la estructura de los datos disponibles otra vez, lo que obligó a adaptar varios programas y crear uno nuevo para procesar este nuevo tipo de datos JSON. Se reorganizó el repositorio, se creo un diccionario para identificar los

partidos, se añadió un Makefile para facilitar la ejecución y limpieza de scripts, y se creó una documentación inicial con README.

- **Febrero:** Se inició un estudio sistemático de las frecuencias asociadas a distintos tipos de eventos, como goles, faltas o celebraciones. También se ajustaron los tiempos de anotación de eventos para que coincidieran con precisión con la señal de audio. Durante esta fase se realizaron pruebas de reducción de ruido con diversas técnicas, buscando una mayor claridad en las grabaciones.
- **Marzo:** Se aumentó la ganancia de los fragmentos de audio seleccionados para estudiar los eventos y se extrajeron características acústicas potencialmente relevantes (como MFCC, espectrogramas, etc.) para poder estudiarlas y evaluar su posible utilidad. Se aplicaron procesos de normalización para homogeneizar las señales y preparar los datos para el modelado. Además, se comenzaron a desarrollar programas para aplicar técnicas de aprendizaje automático con Random Forest y Ada Boost.
- **Abril:** Se entrenaron y evaluaron varios modelos de aprendizaje automático y aprendizaje profundo, entre ellos Gradient Boosting, CNN y LSTM, diseñados para detectar los goles y equipos durante los partidos. Se experimentó con distintas arquitecturas y se afinaron parámetros para mejorar el rendimiento.
- **Mayo:** Finalmente, se procedió a documentar todos los módulos, refactorizar el código, con ayuda de la herramienta Sonnar, para mayor claridad y estabilidad, y estructurar los resultados y análisis obtenidos. Este proceso de cierre incluyó la elaboración de gráficos comparativos, métricas de evaluación y una preparación exhaustiva de la memoria escrita.

4 Diseño y desarrollo

Este proyecto se ha centrado en analizar e identificar eventos de gol que se desarrollan durante un partido de fútbol, procesando para ello la señal de audio capturada siempre en el mismo estadio, Santiago Bernabéu, con el fin de mantener la consistencia en los datos empleados para el entrenamiento del sistema.

A lo largo de este capítulo se detallarán los los distintos componentes del sistema: la arquitectura general, la descripción de los módulos que lo integran, las técnicas de visualización y análisis exploratorio de los datos, la detección de los eventos sonoros relevantes, el proceso de construcción de la base de datos utilizada y la estrategia seguida para la selección e implementación de algoritmos de aprendizaje automático.

Este sistema permite no solo estudiar el comportamiento sonoro durante un evento deportivo, sino también abrir la puerta a aplicaciones futuras como el análisis emocional del ambiente.

4.1. Diseño

4.1.1. Organización del código

```
unican-listentothestadium/
|-- src/
|   |-- constants.py
|   |-- match_mapping.py
|   |-- utilities.py
|   |-- data_preparation/
|   |   |-- audio_analysis.py
|   |   |-- audio_cleaner.py
|   |   |-- Json_Data.py
|   |   |-- search_goals.py
|   |-- goals_classification/
|   |   |-- goals_success.py
|   |   |-- split_goals.py
|   |-- goal_detection/
|   |   |-- goal_detector_ada_boost.py
|   |   |-- goal_detector_lstm_model.py
|   |-- training/
|       |-- ada_boost.py
|       |-- cnn.py
|       |-- cnn_1d.py
|       |-- gradient_boosting.py
|       |-- lstm.py
|       |-- model_comparator.py
|       |-- random_forest.py
|-- scripts/
|   |-- data_process.py
|   |-- train_models.py
|   |-- classificate_goals.py
|   |-- detect_goals.py
|-- requirements.txt
|-- setup.py
|-- Makefile
```

Figura 4.1: Estructura de directorios del proyecto.

4.1.2. Clases

En el proyecto se han definido diversas clases para encapsular funcionalidades y garantizar una arquitectura modular y mantenible. A continuación se describen las principales:

- **constants**: centraliza todas las rutas de ficheros, nombres de carpetas y parámetros globales (por ejemplo, umbrales de detección o constantes de audio). De esta forma, cualquier cambio de configuración solo exige modificar esta clase.
- **utilities**: agrupa funciones auxiliares reutilizables, como cálculo de métricas (precisión, recall), gestión de logs y operaciones comunes de manipulación de arrays o DataFrames. Se importa desde todos los módulos que requieran estas utilidades.

- **data_preparation** (paquete): responsable de la carga y preprocesado de datos brutos. Contiene las siguientes clases:
  - `audio_analysis`
  - `audio_cleaner`
  - `Json_Data`
  - `search_goals`

  Internamente usa `constants` y `utilities` para operaciones estándar.
- **goal_classification** (paquete): una vez detectado un posible gol, clasifica a qué equipo pertenece. Se entrena con datos etiquetados de `data_preparation`. Incluye:

- $\texttt{goal}_s uccess$

- **training** (paquete): orquesta el proceso de entrenamiento y evaluación. Para cada combinación de algoritmo y parámetros:
  1. Llama a `data_preparation` para obtener datos.
  2. Extrae características.
  3. Entrena y valida.
  4. Genera un informe con resultados y métricas.

  Contiene las siguientes clases/script-launchers:
  - `ada_boost`
  - `cnn`
  - `cnn_1d`
  - `gradient_boosting`
  - `lstm`
  - `random_forest`
  - `model_comparator`
- **goal_detection** (paquete): encapsula las estrategias de detección de goles. Cada detector devuelve los instantes de tiempo en que se predice un evento gol:
  - `goal_detector_ada_boost`
  - `goal_detector_lstm_model`
- **Makefile**: programa de más alto nivel que expone un único método `run`, el cual coordina todas las fases anteriores:
  1. Preprocesado de datos (`data_preparation`)
  2. Clasificación de goles (`goal_classificationr`)
  3. Entrenamiento y comparación de modelos (`training`)
  4. Detección de goles (`goal_detection`)

  Gracias a esta clase, la ejecución completa se lanza con un simple comando, facilitando su uso y reproducibilidad.

Con esta organización basada en clases independientes y bien definidas, se consigue una separación clara de responsabilidades, facilidad para pruebas unitarias y escalabilidad para añadir nuevas técnicas de detección o clasificación.

4.1.3. Herramientas y tecnologías

El proyecto se ha desarrollado en Python 3, por su versatilidad en procesamiento de datos y disponibilidad de librerías científicas. A continuación se listan las principales librerías y herramientas utilizadas:

- **Entorno y gestión de dependencias**
  - `virtualenv` / `venv`: creación de entornos aislados.
  - `pip`: instalación de paquetes (fichero `requirements.txt`).
  - `Makefile`: automatización de comandos recurrentes (preprocesado, entrenamiento, evaluación).
- **Procesado de audio y señales**
  - `librosa`: carga de audio, normalización y extracción de MFCC, cromas y energía.
  - `scipy.signal`: filtrado y análisis de señales.
- **Análisis de datos y manipulación**
  - `pandas`: estructuras `DataFrame` para metadatos y resultados.
  - `numpy`: operaciones numéricas y álgebra lineal.
- **Modelado y machine learning**
  - `scikit-learn`: técnicas clásicas (AdaBoost, SVM, validación cruzada, métricas).
  - `TensorFlow` y `Keras`: construcción y entrenamiento de redes LSTM.
  - `joblib`: serialización de modelos entrenados.
- **Visualización y generación de informes**
  - `matplotlib`: generación de gráficos de resultados.
  - `seaborn`: representación gráfica de distribuciones y comparativas.
- **Control de versiones y colaboración**
  - `git`: control de código fuente.
  - `GitLab`: repositorio remoto y gestión de issues.

Con esta selección de herramientas se asegura un flujo de trabajo reproducible, documentación clara y capacidad de escalar o incorporar nuevas técnicas en el futuro.

4.2. Desarrollo

La arquitectura del proyecto esta diseñada de forma modular, para una mayor flexibilidad y escalabilidad del código. El sistema se compone de una serie de clases secuenciales, cada una especializada en una función específica dentro del procesamiento del audio, desde la captura inicial hasta la generación de resultados analíticos. La arquitectura se divide en los siguientes bloques:

- **Adquisición y preparación de los datos:** El sistema recibe como entrada grabaciones de audio en formato .mp4 y archivos JSON que contienen anotaciones temporales de eventos relevantes (goles, faltas, celebraciones, etc.). Todos estos datos se ordenan en carpetas diferenciadas por partido, las cuales, al igual que los archivos, se nombran adecuadamente gracias a un diccionario de partidos. Esta información es importante para vincular lo que sucede en el campo con las señales acústicas correspondientes.

- **Procesamienteo de datos JSON:** Se procesan los archivos JSON para extraer y estructurar los eventos clave. Se transforma la información en un formato que pueda ser utilizado para el análisis temporal y se evita información irrelevante. Esto permite facilitar la correlación entre el audio ambiental y su contexto.
- **Procesamiento del audio:** Se aplican una serie de técnicas para mejorar la calidad del audio y normalizarlo para su posterior análisis. Esto incluye la separación de fuentes para eliminar el comentarista, la limpieza de ruido mediante filtros y la reducción espectral, y la normalización de niveles de amplitud. Este proceso asegura que las características relevantes del sonido ambiente no se pierdan ni se distorsionen.
- **Análisis Exploratorio de Datos (EDA):** Una vez procesado, el audio se somete a una exploración inicial con el fin de visualizar métricas básicas como amplitud, frecuencia, presencia de patrones acústicos y correlaciones temporales. Esto se realiza mediante representaciones gráficas como espectrogramas, histogramas de frecuencia o líneas de tiempo de intensidad sonora.
- **Detección de eventos y generación de base de datos:** Mediante un algoritmo basado en frecuencia y tiempo se identifican los posibles goles que conformarán la base de datos. Se crea la base de datos y se etiquetan los posibles goles indicando si se ha producido realmente y ,en su caso, el equipo marcador. Además, se añaden a cada posible gol las características relevantes para el posterior aprendizaje.
- **Análisis e inferencia:** Una vez que los datos están listos, se aplican algoritmos de apreandizaje automático (Random Forest, Boosting, LSTM, CNN), para clasificar eventos y detectar patrones acústicos sobre el sonido ambiente.
- **Visualización de resultados:** Finalmente, se generan gráficos y visualizaciones que muestran la evolución del ambiente sonoro, la aparición de eventos y los niveles emocionales detectados en distintas fases del partido. Esto permite interpretar los resultados y validar el rendimiento del sistema.

Esta arquitectura modular permite mantener la escalabilidad del sistema y facilita su futura mejora, permitiendo añadir nuevas fuentes de información, nuevas métricas de análisis o modelos más sofisticados a medida que el sistema evolucione.

4.3. Descripción de los módulos

El proyecto como ya se ha explicado se estructura en una arquitectura modular, esta división responde tanto a la funcionalidad del código como la metodología seguida, con un flujo claro desde la obtención de los datos hasta la obtención de resultados visuales y analíticos útiles para la comprensión del audio ambiental.

Cada módulo actúa de manera independiente pero relacionada con los demás, lo que facilita el mantenimiento del código así como su escalabilidad y adaptación a posibles nuevos escenarios. Además, esta separación permite la identificación rápida de errores y una mayor trazabilidad de los datos a lo largo de los diferentes módulos.

4.3.1. Adquisición y preparación de los datos

La primera fase del sistema consiste en la adquisición y preparación de los datos. Esto es fundamental debido a que la calidad y coherencia de la información recopilada impacta en la precisión del posterior análisis. Las dos fuentes principales de datos son los vídeos de los partidos y los

datos estadísticos de los mismos.

Obtención de los vídeos

Los vídeos se obtienen a través de la plataforma Wyscout [1] , ampliamente utilizada en el ámbito del análisis de los partidos de fútbol. Esta plataforma ofrece la posibilidad de descargar las grabaciones completas de los partidos recortando los momentos irrelevantes, como lo son los descansos. Estas grabaciones están en formato MP4, extraídas directamente de retransmisiones televisivas. Esta característica implica que los partidos presentan diferencias significativas en cuanto al entorno sonoro, ya que no proceden de la misma fuente, variando los comentaristas, el volumen del sonido tanto ambiente como el de los micrófonos de los comentaristas y los distintos niveles de mezcla entre los elementos del audio.

Esta heterogeneidad es especialmente relevante en el contexto de este trabajo, centrado en el análisis del audio ambiental. Por tanto, es necesario llevar a cabo posteriormente un proceso de limpieza y normalización del sonido para garantizar la consistencia entre las distintas muestras.

Obtención de los datos estadísticos

Los datos estructurados que describen los eventos del partido se obtienen desde la plataforma StatsBomb [2]. Esta fuente proporciona ficheros en formato JSON que contienen información detallada y temporalizada sobre las acciones del partido (pases, tiros, faltas, goles, etc.). El primer paso consiste en localizar el "match_id" correspondiente al partido deseado, para lo cual se consulta un archivo general que lista todos los encuentros disponibles, junto con sus respectivos identificadores.

En este archivo se realiza un filtrado basado en el equipo local, en este caso "Real Madrid", y una vez localizado el identificador del encuentro, se descarga el fichero específico que contiene los eventos (id_events.json). Este fichero incluye una descripción secuencial y cronológica de todas las acciones ocurridas durante el partido, información que será utilizada posteriormente para sincronizarla con el audio del vídeo y extraer características relevantes.

Organización de los datos

Una vez recopilado el contenido audiovisual y los datos de eventos, se procede a organizar la información en una estructura de carpetas uniforme. Esta estructuración no solo mejora la organización general del proyecto, sino que también permite el acceso automatizado desde scripts y herramientas de análisis posteriores. En concreto, los datos se organizan dentro de una carpeta principal denominada Match_tables. En ella, se crea una subcarpeta por cada encuentro siguiendo la convención Match_RMA-XXX, donde XXX representa una abreviatura del equipo rival.

Dentro de cada carpeta individual se almacenan tanto el vídeo (RMA-XXX.mp4) como el archivo con los datos de eventos (RMA-XXX_all_events.json). Esta nomenclatura homogénea facilita la vinculación directa entre ambos tipos de datos y evita errores durante la carga y procesamiento.

Referencia cruzada

Para facilitar el acceso programático a los datos de cada partido, se implementa un diccionario

[1] https://www.hudl.com/en_gb/products/wyscout
[2] https://www.hudl.com/en_gb/products/statsbomb

denominado match_mapping. Este diccionario actúa como una tabla de referencia cruzada, donde a cada partido se le asigna un número entero como identificador.

De esta forma, se puede acceder a la información mediante un índice numérico, evitando la necesidad de trabajar directamente con cadenas de texto o nombres de archivo. Este mecanismo resulta especialmente útil durante la automatización de tareas de análisis y entrenamiento de modelos, donde se procesan múltiples partidos de forma secuencial.

4.3.2. Procesamiento de los datos JSON

Una vez adquiridos y organizados los datos correspondientes a los partidos, es necesario transformar y estructurar la información contenida en los archivos JSON para que pueda ser analizada de manera eficiente. En esta fase se realiza la lectura, estructuración y preprocesamiento de los datos del evento, convirtiendo el contenido semiestructurado en un formato más manejable (CSV) y añádiendo columnas derivadas que falicitan el posterior análisis.

El primer paso consiste en localizar y cargar el archivo JSON correspondiente al partido, cuyo nombre está determinado por el identificador almacenado en el diccionario "match_mapping" mencionado anteriormente. Una vez cargado el JSON, los datos se convierten en un DataFrame de pandas [3], lo que facilita enormemente su manipulación y análisis.

Una de las tareas principales del procesamiento es la generación de una escala temporal uniforme. A partir de los campos originales minute, second y duration, se calculan dos columnas nuevas: Start y End. Start representa el segundo exacto dentro del partido en que comienza el evento y End marca su finalización.

Dado que la estructura temporal de la segunda mitad puede presentar solapamiento respecto a la primera debido a el tiempo extra añadido por el colegiado, se calcula un desfase temporal (offset) basado en la duración de la primera parte. Luego, se ajustan los eventos de la segunda mitad sumando el offset para asegurar una línea temporal continua. Este paso es crucial para sincronizar con precisión los eventos del JSON con el audio del vídeo, especialmente para tareas como la detección de picos de sonido relacionados con acciones del juego.

Una vez generada la escala temporal, se enriquecen los datos extrayendo información adicional de campos anidados del JSON:

- **Equipo y jugador** Se obtienen de campos anidados dentro del campo de evento.
- **Ubicación** Corresponde a las coordenadas del campo donde ocurre el evento.
- **Tipo de evento** Describe la naturaleza del evento (pase, tiro, falta, etc.).
- **Descripción** Se genera a partir de una lógica condicional personalizada. Según el tipo de evento, se accede a subcampos específicos que permiten extraer detalles como el tipo de pase, si el regate fue exitoso, el resultado del disparo, si hubo presión, entre otros. Esta columna proporciona una mayor especificidad semántica y permite análisis más precisos.

4.3.3. Procesamiento del audio

En esta fase se transforma el vídeo original del partido en una señal de audio limpio y listo tanto para la extracción de características acústicas como para la detección de eventos. El proceso consta de tres subetapas principales: separación de fuentes, filtrado de ruido y normalización del

[3] https://pandas.pydata.org/docs/

audio.

Separación de la voz del comentarista

Como primer paso en el procesamiento del audio, resulta fundamental aislar el ambiente sonoro del estadio de la pista de la narración para evitar que la presencia del comentarista interfiera en el análisis de sentimiento y detección de eventos acústicos.

Se comienza extrayendo la pista de audio del vídeo MP4 original utilizando `ffmpeg`[4], una de las bibliotecas estándar en procesamiento multimedia. Esta herramienta permite generar el archivo `temp_audio.wav`, un fichero en formato WAV sin compresión que conserva el rango dinámico completo del audio grabado, sin pérdida de calidad.

A continuación, se ejecuta el modelo de separación de fuentes Demucs ("htdemucs") entrenada específicamente para descomponer audio musical en pistas elementales (voz, bajo, batería, resto). Aunque Demucs está orientada al ámbito musical, su arquitectura U-Net adaptada a señales de audio la hace idónea para segmentar la narración (voz) de los efectos y el público.

El modelo produce un directorio con las cuatro pistas separadas:

- **bass.wav** Bajo.
- **drums.wav** Percusión.
- **other.wav** Efectos y ambiente.
- **vocals.wav** voz, en este caso el comentarista.

Con la librería Python pydub [5], se cargan las tres pistas que no tienen voz (bass, drums, other) y se superponen secuencialmente para generar un único archivo RMA-XXX.wav. Este procedimiento conserva la coherencia temporal de las pistas y garantiza que los sonidos de baja y alta frecuencia (rúido de la multitud, impactos, cánticos) permanezcan intactos

Esta etapa, basada en Demucs, permite la separación optima de la voz del comentarista para los posteriores pasos de filtrado y análisis del audio.

Filtrado de ruido con filtros FIR de paso-bajo

Tras la eliminación de la voz del comentarista, el audio restante aún puede contener artefactos de grabación y componentes de alta frecuencia, como siseos, interferencias eléctricas, efectos indeseados de compresión o pops del micrófono, que no aportan información relevante al análisis del sonido ambiental. Para atenuar estos elementos sin comprometer las reacciones del público, se recurre al diseño e implementación de un filtro digital FIR (Finite Impulse Response) de paso-bajo. [13]

Un filtro FIR se caracteriza por tener una respuesta al impulso de duración finita, lo que garantiza estabilidad absoluta y un control preciso de la fase. En nuestro caso, se emplea una ventana de Hamming para generar los coeficientes del filtro, con los siguientes parámetros, determinados mediante un proceso iterativo de ajuste:

[4] https://ffmpeg.org/
[5] https://github.com/jiaaro/pydub

- **Número de coeficientes (numtaps):**512. Un mayor número de coeficientes permite transiciones más suaves entre la banda de paso y la de rechazo, reduciendo el "ringing" (oscilaciones no deseadas) en la señal filtrada.
- **Frecuencia de corte:** 4 000 Hz. Se ha seleccionado tras comprobar que la mayoría de las frecuencias relevantes del ruido de público (aplausos, cánticos, gritos) caen por debajo de este umbral, mientras que las interferencias de alta frecuencia quedan fuera de la banda de paso.
- **Ancho de transición:** 100 Hz. Este parámetro define cuán gradual es la atenuación entre los 4 000 Hz útiles y las frecuencias superiores que se desean suprimir; controla la suavidad con la que el filtro pasa del rango de paso al de atenuación.

Una vez diseñados los coeficientes, el filtro se aplica directamente a la señal de audio cruda. Si el audio es estéreo, cada canal se procesa por separado y luego se recombinan, garantizando que la mezcla espacial del sonido no se altere. La operación de filtrado atenúa progresivamente todas las componentes por encima de 4 000 Hz, minimizando los artefactos de alta frecuencia sin introducir distorsiones de fase significativas.

El filtrado puede reducir ligeramente el nivel general de la señal. Para compensarlo, se introduce una ganancia moderada que aumenta la amplitud de las muestras filtradas. A continuación, se recortan todos los valores al rango natural de un archivo PCM de 16 bits (–32 768 a +32 767) para evitar saturación. Este paso asegura que el sonido amplificado conserva su integridad y se mantiene dentro de los límites operativos.

El resultado es un fichero de audio "limpio", generalmente denominado *_clean.wav, que presenta un espectro libre de componentes de alta frecuencia indeseadas.

Normalización y clipping

Tras obtener un audio limpio y filtrado, resulta esencial escalar y limitar la señal para que todos los segmentos de interés se sitúen dentro de un rango uniforme, facilitando comparaciones y asegurando estabilidad numérica en los análisis posteriores.

Es frecuente que ciertos fragmentos del audio presenten picos muy altos que podrían provocar distorsiones o dificultar el análisis. Para evitarlo, se aplica un proceso de soft clipping que suaviza estos picos en lugar de recortarlos de forma brusca. Esto no elimina por completo los picos, pero los suaviza para que la transición sea más natural, evitando la saturación abrupta.

Una vez suavizados los picos, se realiza una normalización global de la señal. El objetivo es escalar todas las muestras para que la amplitud máxima de la señal alcance un valor estándar y uniforme.

Este paso es crucial por varias razones:

- **Comparabilidad entre audios:** permite contrastar partidos distintos aunque se hayan grabado con diferentes micrófonos o configuraciones.
- **Consistencia en umbrales:** algoritmos posteriores que dependen de valores relativos (por ejemplo, detección de picos por encima de cierto nivel) se benefician de una escala homogénea.

- **Prevención de saturación:** al asegurar que el valor máximo es conocido y controlado, se evita cualquier tipo de clipping accidental posterior.

La función normalize_audio, desarrollada específicamente para este proyecto, se encarga de este proceso. Divide todas las muestras por el valor absoluto máximo de la señal (después del clipping) y luego las multiplica por el valor objetivo. Esto mantiene la forma relativa de la onda, pero asegura que el volumen general sea adecuado y consistente entre diferentes archivos.

Para evitar que el resultado final quede demasiado bajo o excesivamente alto, se mide la amplitud media del audio normalizado. En función de su valor, se repite el proceso de amplificación, clipping y normalización con parámetros distintos.

Este proceso adaptativo puede activar dos casos:

- Si la amplitud media es demasiado baja (< 925), se vuelve a amplificar con un factor_interval mayor y se usa un threshold más bajo para el clipping.
- Si la amplitud media es demasiado alta (> 1200), se atenúa el volumen y se permite un margen mayor en el clipping, lo que reduce la posibilidad de distorsión.

El audio final se guarda en un archivo .wav ya amplificado, suavizado y normalizado, listo para análisis posteriores o entrenamiento de modelos. En paralelo, se genera una tabla CSV que contiene el momento donde se producen los goles, la amplitud media por segundo del evento, su duración, su pico máximo y la amplitud anterior a este, facilitando el posterior análisis de estos eventos. Un ejemplo de la tabla resultante puede verse en la Figura 4.2

		1	Start_S…	Amplitude_Arr…	Previous_Amplitude	Maximum_Amplitude		Mean_Consecutive_Amplitude	
1	1	Real …	569	[np.float64(0.270468…	0.2732009887695312	0.5228200739080255	16	0.4647165731950239	…
2	1	Real …	1159	[np.float64(0.320707…	0.2339517766779119	0.4948842828924006	15	0.4316790031664299	…
3	1	Real …	1985	[np.float64(0.441782…	<null>	0.530571816184304	21	0.4810203915550595	…
4	1	Real …	3307	[np.float64(0.430338…	0.2632619684392756	0.45347900390625	6	0.3480002894546046	…
5	1	visit…	2080	[np.float64(0.462842…	0.2917613636363637	0.5028890436345881	16	0.4345180511474609	…
6	1	visit…	5277	[np.float64(0.248599…	0.281309647993608	0.4499880704012784	4	0.4167567513205788	…
7	2	visit…	27	[np.float64(0.227613…	0.1915323430841619	0.4741471724076705	14	0.4279678245643516	…
8	2	Real …	1057	[np.float64(0.105836…	0.1413836392489346	0.4880857987837357	15	0.4589840050899622	…
9	2	Real …	1938	[np.float64(0.288166…	0.2881661155007102	0.5064215920188211	19	0.4601091229744505	…

Figura 4.2: Extracto de una tabla que recoge los datos sonoros de cada uno de los goles.

4.3.4. Análisis Exploratorio de Datos (EDA)

El análisis exploratorio de datos (EDA) es una fase previa al modelado que permite conocer en profundidad las características de la señal de audio y su relación con los eventos deportivos anotados. A través de visualizaciones y estadísticas básicas, el EDA facilita la identificación de patrones relevantes, la detección de anomalías y la validación de hipótesis sobre la reacción del público.

En primer lugar, se realiza la carga del archivo de audio final utilizando librerías como pydub y librosa[6], lo cual permite extraer la frecuencia de muestreo, la duración total y las muestras en formato monofónico. Esta operación asegura la correcta lectura de la señal y sienta las bases para las transformaciones posteriores.

[6] https://librosa.org/

A continuación, se representa la forma de onda completa de la grabación. Dado que el audio puede contener cientos de miles de muestras, se ha reducido su resolución mediante un "down-sampling" sistemático. Sobre el eje temporal resultante se superponen rectángulos que indican los intervalos de los eventos (goles, faltas, etc.), distinguiendo los distintos equipos por color. Esto permite observar que los picos de amplitud coinciden con los momentos en los que los aficionados reaccionan a las jugadas, validando así que la amplificación y el filtrado previos han preservado la dinámica relevante como se muestra en la Figura 4.3.

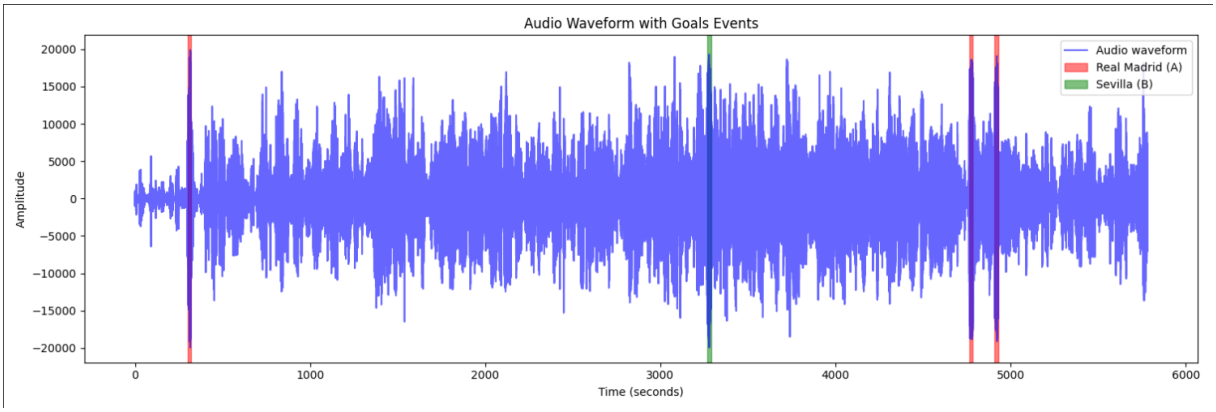

Figura 4.3: Waveform de partido (Real Madrid - Sevilla) con eventos de gol representados.

Para profundizar en como varía la energía acústica, se divide la señal en fragmentos de tamaño fijo y se calcula la suma de cuadrados en cada uno. Estos valores se convierten en decibelios y se traza la curva de energía en dB respecto al tiempo, nuevamente superponiendo los intervalos de eventos. Al visualizar el audio en esta escala, Figura 4.4, confirmamos que las acciones más destacadas generan incrementos claros en la energía.

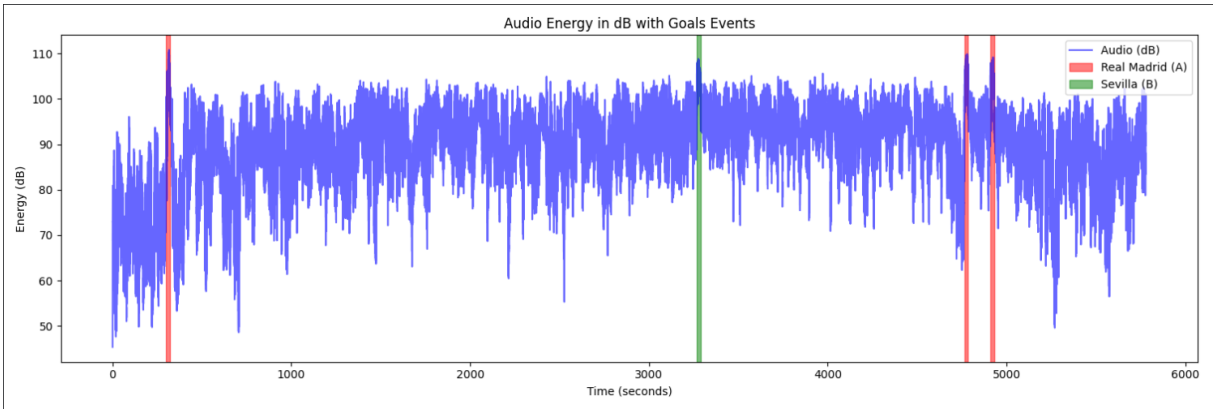

Figura 4.4: Gráfica de energía (dB) de partido (Real Madrid - Sevilla) con eventos de gol representados.

El análisis espectral proporciona una visión más detallada de la distribución de frecuencias en cada intervalo de evento. Para ello, se extrae el fragmento de audio correspondiente, se le aplica una Transformada Rápida de Fourier (FFT) y se calcula el nivel en decibelios por frecuencia. Posteriormente, se promedia la energía de los picos más prominentes y se determina la frecuencia de resonancia como la mediana de las tres frecuencias consecutivas con mayor energía. Este en-

foque permite identificar patrones comunes entre tipos de evento, como resonancias dominantes en el rango de 200 a 600 Hz durante los goles. Un ejemplo de este análisis espectral se muestra en la Figura 4.5

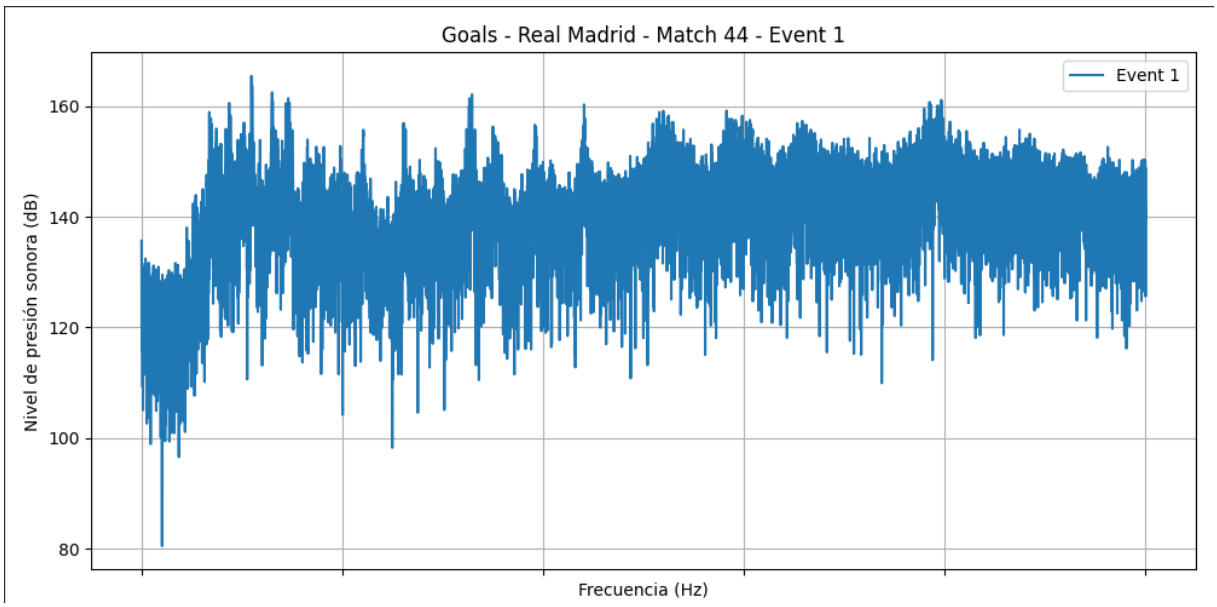

Figura 4.5: Gráfica de frecuencia de gol del Real Madrid en partido(Real Madrid - Sevilla).

Se ha explorado también una segmentación acústica del partido basada en bloques de tiempo fijo (por ejemplo, 10 segundos), donde se calcula el centroide espectral de cada segmento. Esta métrica, que resume la "gravedad.º "agudeza"del contenido espectral, se ha utilizado para asignar eventos a equipos, asumiendo que las distintas aficiones generan firmas acústicas diferentes para estudiar si el tono sería también un factor importante de diferenciación.
Por último, se ha representado la forma de onda de cada uno de los goles en ventanas de tiempo que van aumentando de tamaño, esto se ha hecho para poder observar si el intervalo está posicionado correctamente y también para saber cuando somos capaces de diferenciar o no los distintos goles simplemente observando la forma de onda.

En conjunto, este análisis exploratorio permite confirmar que existe una sincronización adecuada entre las anotaciones manuales y la señal sonora, lo cual es fundamental para garantizar la coherencia del estudio. Además, facilita la identificación de correlaciones claras entre distintas características acústicas y los tipos de eventos registrados durante el partido, como goles, faltas o tiros libres. A partir de estos hallazgos, se pueden ajustar con mayor precisión los parámetros empleados tanto en el análisis como en la detección automática, optimizando así el rendimiento en las siguientes etapas del proyecto.

4.3.5. Detección de eventos y generación de base de datos

La generación de la base de datos de entrenamiento parte de la identificación automática de "ventanas" de audio donde la energía es sustancialmente mayor que en el resto del partido, criterio que suele corresponder con goles u otras jugadas de gran repercusión sonora. Para ello, se emplea un algoritmo de detección basado en la envolvente de amplitud suavizada.

Se carga el audio de un partido y se calcula la envolvente de amplitud por tramos (frames) superpuestos. Esta envolvente se filtra con una media móvil para atenuar fluctuaciones breves y

facilitar la detección de segmentos continuos de alta energía. Cada vez que la envolvente supera un umbral mínimo y se mantiene por un tiempo mayor al definido (por ejemplo, 12 s), se marca el inicio y el final de un intervalo de interés. Para evitar detecciones contiguas excesivamente próximas, se exige un espacio temporal mínimo entre intervalos consecutivos.

Una vez identificados estos intervalos, se procede a extraer un conjunto de características acústicas de cada uno. Sobre el fragmento de muestras correspondiente, se calcula:

- La serie de amplitudes en dB por ventanas fijas de corta duración.
- La frecuencia de resonancia, determinada a partir de la FFT como la frecuencia con mayor magnitud, que suele reflejar la región espectral dominante en la reacción del público.
- Estadísticas de energía (media, varianza, crest factor) que resumen la dinámica del tramo.
- El centroide espectral, indicador de si el espectro se desplaza hacia frecuencias más agudas o más graves durante el evento.

Estas características se almacenan en un CSV global, donde cada fila corresponde a un intervalo detectado e incluye columnas tales como número de partido, equipo, instante de inicio, niveles de amplitud, frecuencia de resonancia y etiquetas de éxito de gol (inicialmente vacías).

Para completar la asignación de etiquetas, se fusiona esta tabla de características con la tabla de evaluación de eventos, que contiene la información de los goles reales. Se cruzan ambos conjuntos por número de partido y tiempo de inicio, asignando a cada fila una columna Goal_Success con valor 1 (gol) o 0 (no gol) y el resultado se guarda, como se puede observar en la Figura 4.6.

	…	T…	C3	C4	C5	C6	C7	C8	C9	C10
1	…	T…	Goal_Ampl…	Goal_Resonant_F…	Goal_Avg…	Goal_Dom…	Goal_Amplitude_Var…	Goal_Crest…	Goal_Spectral_Cen…	Goal_Suc…
2	1	R…	[np.float…	106.0	14081030…	106.0	14080987.0	4.54	1193.5	1
3	1	R…	[np.float…	105.75	18994312…	105.75	18992656.0	4.5	970.84	1
4	1	R…	[np.float…	102.5	14036171…	102.5	14036121.0	4.64	1012.72	1
5	1	V…	[np.float…	665.58	15273100…	665.58	15273088.0	4.56	942.96	1
6	1	R…	[np.float…	102.5	12211445…	102.5	12211371.0	5.45	1053.46	1
7	1	<…	[np.float…	410.25	7835408.5	410.25	7835343.5	6.6	1467.62	0
8	1	V…	[np.float…	414.0	11158623…	414.0	11158338.0	5.09	1020.96	1

Figura 4.6: Extracto del conjunto de datos que se utilizará para los modelos.

Finalmente, para preparar un conjunto de datos de audio que sirva directamente como entrada a la red LSTM, se recorre goals_features.csv partido por partido. Para cada intervalo etiquetado, se extrae un segmento de audio de 20 s a partir del instante de inicio (empleando librosa y soundfile), y se guarda en dos carpetas separadas: Audio_goles/gol y Audio_goles/no_gol, según la etiqueta de éxito. De esta manera, se obtiene un dataset balanceado de clips “gol” y “no gol” que permitirá entrenar la LSTM usando directamente las muestras de audio.

Este proceso automatizado asegura que cada fragmento de audio seleccionado refleje de forma fiel la dinámica sonora asociada a eventos de alto interés, y proporciona tanto las características numéricas como los propios ficheros WAV necesarios para la fase de modelado y validación de la detección de goles mediante técnicas de aprendizaje.

4.3.6. Análisis e inferencia

En esta etapa, las características extraídas de cada fragmento de audio, como los fragmentos en si, se emplean para entrenar y evaluar distintos modelos de aprendizaje automático, con el fin de detectar automáticamente dos tareas principales: determinar si un intervalo corresponde a un gol y una vez detectados estos goles, asignar el gol al equipo correcto.

En primer lugar, se codifica la columna Team como variable numérica (Real Madrid = 1, Visitante = 0, sin asignación = -1), y se eliminan campos irrelevantes para el modelo, como identificadores de partido o tiempo exacto de inicio del evento. Todas las columnas que contienen listas de valores (por ejemplo, secuencias de amplitud en dB) se convierten a un único valor representativo (la media), transformando así los datos en un formato de baja dimensionalidad.

A continuación, el conjunto de datos se divide en tres subconjuntos: entrenamiento (60 % del total), validación (20 %) y prueba final (20 %). Sobre los datos de entrenamiento, se entrenan un modelo Random Forest, un AdaBoost y un Gradient Boosting, con el objetivo de determinar qué intervalos corresponden a un gol. El mismo procedimiento se repite para entrenar un segundo Random Forest, AdaBoost y Gradient Boosting, destinados a predecir la etiqueta "Team" únicamente en los casos previamente clasificados como gol.

Por otro lado, el conjunto de fragmentos de audio obtenidos anteriormente también se divide en los mismos tres subconjuntos. Sobre los fragmentos de entrenamiento, se entrenan una CNN y una LSTM, también con el objetivo de determinar qué fragmentos corresponden a un gol.

4.3.7. Visualización de resultados

En la fase de visualización de resultados, se parte de la tabla results.csv, que aglutina para cada modelo entrenado sus métricas más relevantes: exactitud en validación y prueba (val_accuracy, test_accuracy), así como precisión, recall y F1-score.

En una figura, que se mostrará más adelante, se emplea un diagrama de barras que superpone la exactitud obtenida en el conjunto de validación y en el conjunto de prueba para cada uno de los modelos evaluados. Este gráfico permite comparar de un vistazo la estabilidad del rendimiento entre fases de entrenamiento y su capacidad de generalización.

Estas visualizaciones ofrecen una visión inmediata de los puntos fuertes y débiles de cada aproximación, sirviendo de guía para decidir qué modelo profundizar o ajustar.

5 Implementación de los modelos

Este capítulo detalla la puesta en práctica de los algoritmos seleccionados para la detección de eventos de gol y la clasificación de equipo a partir del audio ambiental, lo que es la finalidad a la que se pretende llegar. Tras haber presentado en el capítulo 2.1 los modelos basados en árboles (Random Forest, AdaBoost, Gradient Boosting) y las redes neuronales (CNN sobre MFCC y LSTM), en esta sección se describen los aspectos de su implementación, para poder escoger cual de ellos logra un mejor resultado.

5.1. Random Forest

El primer paso consiste en preparar el conjunto de datos para el entrenamiento del Random Forest. Tras cargar el fichero goals_features.csv, se detectan aquellas columnas que contienen listas de valores (por ejemplo, las secuencias de amplitud en dB) y se convierten en escalas numéricas únicas mediante el cálculo de su media. Esta transformación asegura que cada característica pase a ocupar una única columna continua. A continuación, la variable categórica Team se traduce a 1 (Real Madrid), 0 (visitante) o –1 (sin asignación), y se eliminan las columnas auxiliares que no aportan información predictiva, como identificadores de partido o marcas de tiempo exactas.

Para la partición de los datos se sigue un procedimiento reproducible y controlado, utilizando la función train_test_split de scikit-learn en dos etapas. Primero, el 20 % del conjunto total se separa como conjunto de prueba final, de modo que nunca intervenga en el ajuste de ningún modelo ni en la selección de sus parámetros. El 80 % restante se somete a un segundo corte, reservando el 25 % de ese bloque (equivalente al 20 % del total) como conjunto de validación, y destinando el resto (60 % del total) al entrenamiento.

El Random Forest configurado consta de 100 árboles de decisión independientes que se construyen mediante bootstrap sampling: cada uno recibe como entrada una muestra aleatoria con reemplazo de las instancias de entrenamiento, de manera que aproximadamente el 63 % de los ejemplos aparecen al menos una vez en cada árbol.
Este doble mecanismo de aleatorización (muestras de datos + subespacios de variables) pretende lograr dos efectos complementarios:

- **Reducción de varianza:** al promediar las predicciones de árboles entrenados sobre diferentes subconjuntos de datos y características, se amortiguan las oscilaciones extremas que un único árbol podría sufrir ante ruido o valores atípicos del audio.
- **Control del sobreajuste:** mientras que un árbol aislado tiende a ajustarse de forma excesiva al conjunto de entrenamiento (baja sesgo, alta varianza), el bosque aleatorio combina modelos débiles y evita que ninguno domine con sus particularidades, mejorando la robustez global.

Una vez finalizado el entrenamiento, la predicción consiste simplemente en aplicar cada uno de los 100 árboles al vector de características de un intervalo de audio y obtener la clase mayoritaria

por votación simple.

Una vez ajustado el modelo, se mide su exactitud tanto en validación como en prueba. Además de la precisión global, se analizan precisión, recall y puntuación F1, y se visualiza la matriz de confusión con un mapa de calor, lo que permite identificar fácilmente falsos positivos (segmentos no gol clasificados como gol) y falsos negativos (goles no detectados).

Inmediatamente después, se repite el mismo proceso pero focalizado únicamente en los casos etiquetados como gol: se extrae el subconjunto con Goal_Success == 1, se vuelve a particionar en entrenamiento, validación y prueba, y se entrena un segundo Random Forest cuya misión es predecir la etiqueta Team. Este paso permite evaluar no solo la capacidad de distinguir goles de no goles, sino también la habilidad de asignar correctamente el autor del gol en función de las métricas acústicas.

Por último, para facilitar la comparación de este clasificador frente a otros algoritmos, las métricas clave de validación y prueba, test accuracy, val accuracy, precision, recall y F1, se registran en un fichero results.csv. Se añade una fila nueva a este historial bajo el nombre "RandomForest", sentando así las bases para un análisis comparativo posterior.

5.2. AdaBoost

En la implementación del AdaBoost, los datos se preparan de forma idéntica al caso del Random Forest: primero se carga el fichero goals_features.csv, se convierten a escalares aquellas columnas que contienen cadenas con listas de valores (calculando su media) y se recodifica la variable Team a valores numéricos (1 = Real Madrid, 0 = visitante, –1 cuando no se aplica). Tras eliminar las columnas auxiliares (Match_Number, Goal_Start_Time[S], Goal_Time[S]), el conjunto se divide en un 60 % para entrenamiento, un 20 % para validación y un 20 % para prueba final, garantizando así una evaluación rigurosa del rendimiento.

En el script se utiliza `AdaBoostClassifier` con $T = 100$ estimadores y `learning_rate`=1.0. Cada stump entrenado atiende particularmente a los ejemplos mal clasificados en iteraciones previas, lo que permite corregir sus errores y reducir de forma iterativa el sesgo del ensemble sin incurrir en un sobreajuste excesivo.

Una vez entrenado el primer modelo sobre el objetivo Goal_Success, se evalúa en el conjunto de validación y, finalmente, en el de prueba, donde se obtienen la exactitud y el informe de clasificación completo (precision, recall, F1-score). La matriz de confusión resultante se dibuja con un mapa de calor, lo que permite visualizar la proporción de verdaderos positivos, falsos positivos, falsos negativos y verdaderos negativos.

Acto seguido, el mismo flujo se aplica al subconjunto de instancias clasificadas como gol, para entrenar un segundo AdaBoost cuya etiqueta de salida es la variable Team.

Una vez confirmada la estabilidad de ambos modelos en sus respectivos conjuntos de validación y prueba, se aprovecha toda la información disponible para construir las versiones finales de cada clasificador. Para ello se combinan de nuevo todas las muestras de entrenamiento y validación en un único bloque, de modo que cada modelo se reentrene ahora sobre la totalidad de los datos correspondientes a su tarea. Esta práctica, habitual en proyectos de machine learning, busca maximizar la cantidad de ejemplos que ve cada modelo antes del despliegue, mejorando potencialmente su capacidad de generalización.

Los objetos resultantes de estos entrenamientos completos se guardan en disco utilizando la función `joblib.dump`. Así se generan dos ficheros: ada_boost_gol.pkl, que contiene el AdaBoost entrenado para predecir Goal_Success y ada_boost_equipo.pkl, que encapsula el AdaBoost destinado a clasificar la variable Team.

Estos archivos serializados permiten recargar los modelos posteriormente sin necesidad de volver a entrenarlos desde cero, facilitando su uso en entornos de producción o en pruebas posteriores de evaluación.

Finalmente, las métricas clave, exactitud en validación y prueba, precision, recall y F1, se recogen en un diccionario junto con el nombre del modelo ("AdaBoost") y se anexan al fichero results.csv. De esta forma, queda registrada de manera homogénea AdaBoost al conjunto comparativo de clasificadores.

5.3. Gradient Boosting

La implementación de Gradient Boosting reutiliza casi en su totalidad la preparación de datos desarrollado para AdaBoost, pero cambia el algoritmo de ensamblado por uno basado en optimización de gradiente.

Al igual que antes, el conjunto completo se divide en tres particiones, 60 % entrenamiento, 20 % validación y 20 % prueba, garantizando que los datos de evaluación final queden libres de cualquier influencia de ajuste.

Una vez entrenada la primera instancia del modelo para predecir Goal_Success, se evalúa en validación y prueba calculando exactitud, precisión, recall y F1-score, y mostrando la matriz de confusión con un mapa de calor. Seguidamente, se filtra el conjunto a sólo aquellos intervalos etiquetados como gol y se repite el mismo flujo para construir un segundo GradientBoosting-Classifier cuya etiqueta de salida es Team.
Para aprovechar al máximo los datos, se vuelve a reentrenar cada modelo con todas las instancias disponibles de su tarea (entrenamiento + validación originales) y se serializan en disco mediante joblib.dump, generando gradient_gol.pkl y gradient_equipo.pkl. Finalmente, las métricas clave de validación y prueba se extraen de nuevo y se anexan a results.csv bajo la etiqueta "GradientBoosting", lo que permite comparar directamente su desempeño con el de Random Forest y AdaBoost en las visualizaciones y análisis posteriores.

5.4. Red Neuronal Convolucional

A continuación se describe el flujo de trabajo completo para el entrenamiento y evaluación de la red convolucional (CNN) basada en coeficientes MFCC, cuyo objetivo es clasificar fragmentos de audio como "gol" o "no_gol".

5.4.1. Extracción y preprocesado de características

En el preprocesado de audio para la CNN, cada fragmento se convierte en un "mapa espectro–temporal" de características siguiendo estos pasos:

1. **Muestreo y extracción de MFCC.** La señal se carga con librosa a una frecuencia de muestreo de 22050 Hz, suficiente para capturar las frecuencias relevantes de voces y gritos. A continuación se calculan $N_{\mathrm{MFCC}} = 40$ coeficientes MFCC, que capturan la envolvente espectral de corto plazo del audio, este número de coeficientes ofrecen un bien compromiso entre detalle espectral y complejidad computacional [18] .

2. **Cálculo de derivadas.** Para incorporar información sobre la dinámica temporal, se derivan esos MFCC dos veces. Cada derivada añade otros 40 coeficientes, dando un total de $3 \times 40 = 120$ características por frame.

3. **Construcción de la matriz espectro–temporal.** Se apilan verticalmente MFCC, delta y delta2 para formar una matriz de tamaño $120 \times T$, donde T es el número de frames resultantes del muestreo temporal de la señal.

4. **Ajuste de longitud a 100 frames.** Para uniformar la entrada a la red, se fuerza $T = 100$ de manera que si $T < 100$, se rellenan con ceros$y$ si $T > 100$, se truncan los excedentes. El resultado es siempre una matriz de dimensión 120×100. Este número de frames se ha escogido debido a que cubren la duración típica de un gol sin aumentar excesivamente la entrada.

5. **Transposición y canal único.** Para adaptarse a la convolución 2D, se transpone la matriz y se añade una dimensión de canal: $(120 \times 100)^\top \rightarrow (100 \times 120 \times 1)$.

De este modo, cada clip de audio queda representado como un array $(100, 120, 1)$, que contiene simultáneamente información espectral y dinámica temporal, listo para ser procesado por la arquitectura convolucional.

5.4.2. Partición y balanceo

Las siguientes etapas preparan los datos para el entrenamiento del modelo. Primero, las etiquetas textuales `'gol'` y `'no_gol'` se codifican numéricamente mediante la clase `LabelEncoder`, generando un vector binario $y_{\text{encoded}} \in \{0, 1\}^{n_{\text{muestras}}}$ que puede ser interpretado por los modelos de aprendizaje automático. Posteriormente, el conjunto total se divide en subconjuntos de entrenamiento y prueba usando `train_test_split`, reservando el 80 % de los ejemplos para entrenamiento y el 20 % restante para validación. Esta partición asegura que el modelo pueda evaluarse objetivamente sobre datos no vistos durante el entrenamiento.

Dado que podría existir un desequilibrio entre las clases (por ejemplo, más audios sin gol que con gol), se computan pesos de clase utilizando la función `class_weight.compute_class_weight`, que asigna a cada clase un peso inversamente proporcional a su frecuencia en el conjunto de entrenamiento. Esto produce un diccionario de la forma $\{$`0:` w_0`, 1:` $w_1\}$, donde w_0 y w_1 son los pesos para las clases codificadas como `no_gol` y `gol`, respectivamente. Estos pesos se incorporan durante el entrenamiento del modelo para penalizar más los errores cometidos sobre la clase minoritaria, favoreciendo así un aprendizaje más equilibrado.

5.4.3. Definición de la arquitectura CNN

La red neuronal convolucional (CNN) que se usa en este proyecto se construye con la API secuencial de Keras, una herramienta que permite ir añadiendo capas una por una. La estructura comienza con una capa que aplica 32 filtros de tamaño 3×3 a la imagen de entrada (que en este caso representa el audio), el tamaño de los filtros favorece a la detección de patrones locales y el número de filtros permite la captura de características simples [19]. Cada filtro actúa como una pequeña lupa que recorre la imagen detectando patrones. Esta capa usa la función de activación ReLU, que ayuda a que el modelo aprenda de forma más eficiente. Luego se aplica una técnica llamada normalización por lotes (`BatchNormalization`) que estabiliza el aprendizaje. Después, una operación de reducción llamada `MaxPooling2D` toma solo la parte más importante de cada grupo de valores, ayudando a simplificar la información.

El siguiente paso es muy parecido, pero ahora se usan 64 filtros en lugar de 32, lo que permite detectar detalles aún más complejos en la señal de audio transformada [19]. Después de

estos bloques, la información que sale todavía tiene forma de imagen, así que se "aplana" (con `Flatten`) para convertirla en un vector, es decir, una fila de números.

Ese vector se pasa a una capa densa con 64 neuronas, que también usa ReLU para aprender relaciones entre los datos. A esta capa se le aplica un 50 % de `Dropout`, práctica estándar para datasets medianos, una técnica que apaga al azar algunas neuronas durante el entrenamiento para evitar que el modelo memorice en lugar de aprender patrones útiles. Finalmente, la última capa tiene solo una neurona con activación sigmoide, que devuelve un número entre 0 y 1 representando la probabilidad de que el audio contenga un gol.

El modelo se entrena usando un optimizador llamado Adam, con una tasa de aprendizaje muy pequeña (10^{-4}) para que el aprendizaje sea lento pero más preciso. La función que se usa para calcular los errores del modelo es la entropía binaria cruzada (`binary_crossentropy`), especialmente diseñada para este tipo de problemas donde solo hay dos clases posibles "gol" o "no gol".

5.4.4. Control de entrenamiento

Para garantizar un ajuste robusto, se configuran tres callbacks:

- **ReduceLROnPlateau:** reduce la tasa de aprendizaje a la mitad cuando la pérdida de validación no mejore durante 3 épocas.
- **EarlyStopping:** detiene el entrenamiento si la pérdida de validación no mejora en 6 épocas consecutivas, restaurando los pesos de la mejor época.
- **ModelCheckpoint:** guarda en disco el modelo (modelo_cnn_mfcc.keras) con la menor pérdida de validación.

El entrenamiento se ejecuta durante hasta 35 épocas con lotes de 8 muestras, validando contra el conjunto de prueba para monitorizar métricas en tiempo real. El batch pequeño favorece regularización y el límite de 35 épocas cubre convergencia sin sobreentrenar

5.4.5. Evaluación y persistencia

Al finalizar, se predice sobre el conjunto de prueba para calcular accuracy, precision, recall y F1-score, y se muestra un informe detallado (classification_report) junto con la matriz de confusión. Estos resultados se añaden a results.csv bajo la etiqueta "CNN" para su comparación con los demás modelos. Finalmente, se representa gráficamente la evolución de la exactitud en entrenamiento y validación a lo largo de las épocas, lo que permite inspeccionar visualmente el comportamiento de la red durante el ajuste.

Con este flujo, la CNN basada en MFCCs aprovecha tanto la información espectral como las dinámicas temporales del audio para aprender de forma supervisada las señales acústicas que caracterizan un gol en un estadio.

5.5. Long Short-Term Memory

En lugar de aplicar redes convolucionales, en este enfoque se emplea una red basada en LSTM (Long Short-Term Memory), un tipo especial de red neuronal recurrente (RNN) diseñada para trabajar con secuencias. Este tipo de arquitectura resulta especialmente útil en tareas donde el orden y la dependencia temporal entre los elementos es crucial, como en señales de audio.

5.5.1. Extracción y preprocesado de características

El primer paso consiste en cargar los archivos de audio crudo y transformarlos en una representación adecuada para la red neuronal. Utilizando librosa, se cargan los archivos de audio con una frecuencia de muestreo de 22050 Hz y una duración fija de 20 segundos. Si la duración del archivo es menor a esta, se rellena con ceros; si es mayor, se recorta a los primeros 20 segundos. Los archivos de audio son almacenados como vectores 1D de amplitudes en una secuencia temporal, representando el sonido de forma continua.

5.5.2. Construcción del dataset

Con la función load_data, se recorre la carpeta de "gol" y "no_gol", cargando cada archivo de audio crudo y asignando las etiquetas correspondientes. Las etiquetas de cada archivo se codifican con LabelEncoder en valores enteros (0 para "no_gol" y 1 para "gol"). Este conjunto de datos se divide en conjuntos de entrenamiento y prueba, utilizando train_test_split.

5.5.3. Partición y balanceo

El conjunto de datos se divide en un 80 % para entrenamiento y un 20 % para prueba. Dado que los datos pueden estar desequilibrados (más muestras de "no_gol"), se calculan pesos de clase utilizando la función class_weight.compute_class_weight. Estos pesos se pasan al método fit de Keras para garantizar que la red aprenda de manera equitativa las dos clases.

5.5.4. Definición de la arquitectura LSTM

La red neuronal se define utilizando la API *Sequential* de Keras y está diseñada para procesar directamente las secuencias de audio crudo. La arquitectura emplea capas **LSTM (Long Short-Term Memory)** por su capacidad para capturar dependencias temporales en secuencias largas, como las características de la señal de audio a lo largo del tiempo.

- **Entrada:** El modelo recibe como entrada vectores de audio crudo de 20 segundos de duración, muestreados a 22 050 Hz. Cada muestra representa la amplitud del sonido en un instante temporal.
- **Primera capa LSTM:** Se incluye una capa LSTM con 128 unidades, este número de unidades balancea la capacidad de representación y costo, configurada con `return_sequences=True` para que cada instante de tiempo produzca una salida. Esto permite a la siguiente capa LSTM procesar la secuencia completa y conservar la dinámica temporal completa.
- **Dropout:** Para mitigar el sobreajuste, se introduce una capa `Dropout(0.5)`, que desconecta aleatoriamente el 50 % de las neuronas durante el entrenamiento. Esta técnica mejora la generalización del modelo.
- **Segunda capa LSTM:** A continuación, se añade una segunda capa LSTM con 64 unidades y `return_sequences=False`, de forma que solo se retiene la última salida de la secuencia procesada. Esta salida encapsula la representación contextual final del fragmento de audio.
- **Capa densa intermedia:** Posteriormente, se utiliza una capa densa de 64 neuronas con función de activación `ReLU`, la cual transforma la representación temporal en un espacio de características de mayor abstracción.
- **Capa de salida:** Finalmente, se incorpora una capa densa con una única neurona y activación `sigmoid`, que devuelve la probabilidad de que el fragmento de audio represente un evento de "gol".

El modelo se entrena utilizando el optimizador **Adam** con una tasa de aprendizaje de 1×10^{-4} y la función de pérdida `binary_crossentropy`, adecuada para tareas de clasificación binaria. La métrica de evaluación principal durante el entrenamiento es la `accuracy`.
Este diseño permite capturar tanto patrones temporales a corto como a largo plazo presentes en los datos de audio, lo que es crucial para identificar eventos acústicos característicos como los gritos de gol, que pueden variar en duración, tono e intensidad.

5.5.5. Control del entrenamiento

Para garantizar un ajuste robusto, se configuran tres callbacks:

- **ReduceLROnPlateau:** reduce la tasa de aprendizaje a la mitad cuando la pérdida de validación no mejore durante 3 épocas.
- **EarlyStopping:** detiene el entrenamiento si la pérdida de validación no mejora en 6 épocas consecutivas, restaurando los pesos de la mejor época.
- **ModelCheckpoint:** guarda en disco el modelo (modelo_lstm_audio.keras) con la menor pérdida de validación.

El entrenamiento se ejecuta durante hasta 35 épocas con lotes de 8 muestras, validando contra el conjunto de prueba para monitorizar métricas en tiempo real.

5.5.6. Evaluación y persistencia

Una vez entrenado el modelo, se realiza una predicción sobre el conjunto de prueba y se calculan las métricas de rendimiento como la exactitud, precisión, recall y F1-score. Estos resultados se guardan en un archivo CSV para su análisis posterior. Además, se genera un informe detallado mediante classification_report y se muestra la matriz de confusión para evaluar la calidad del modelo.

Finalmente, se representa gráficamente la evolución de la exactitud durante el entrenamiento y la validación a lo largo de las épocas para visualizar cómo se comporta el modelo a medida que se entrena.

6 Análisis de resultados

El principal objetivo del sistema desarrollado ha sido la identificación automática de goles a partir del audio recogido en partidos de fútbol. Para ello, se diseñó un flujo de trabajo que incluye la limpieza del audio, la extracción de características y el entrenamiento de modelos de aprendizaje automático. En esta sección se detallan los resultados obtenidos en esta tarea de detección, atendiendo tanto al rendimiento cuantitativo de los modelos como a los factores que han influido en dicho rendimiento.

Este análisis se centra tanto en el rendimiento de los modelos de clasificación de eventos como en la utilidad de las representaciones gráficas y sonoras generadas durante el proceso.

6.1. Resultados cuantitativos y métricas

Con el fin de evaluar de manera rigurosa el desempeño de los modelos diseñados para la detección de goles en el audio de partidos de fútbol, se recurrió a un conjunto de métricas estándar en problemas de clasificación presentados en la sección 2.1.6. Estas métricas no solo permiten cuantificar el nivel de acierto global del sistema, sino también analizar en detalle la capacidad de cada modelo para identificar eventos reales sin generar predicciones erróneas. A continuación, se presentan los resultados obtenidos en el conjunto de prueba y se realiza un análisis crítico de los mismos.

6.1.1. Resultados obtenidos en el conjunto de prueba

Para cuantificar el rendimiento final de cada modelo, se evaluaron los cinco algoritmos propuestos sobre el conjunto de prueba, que representa el 20 % de los datos totales. En la gráfica se recogen las métricas de exactitud, precisión, recall y F1-score obtenidas para cada modelo, se muestran en la Figura 6.1:

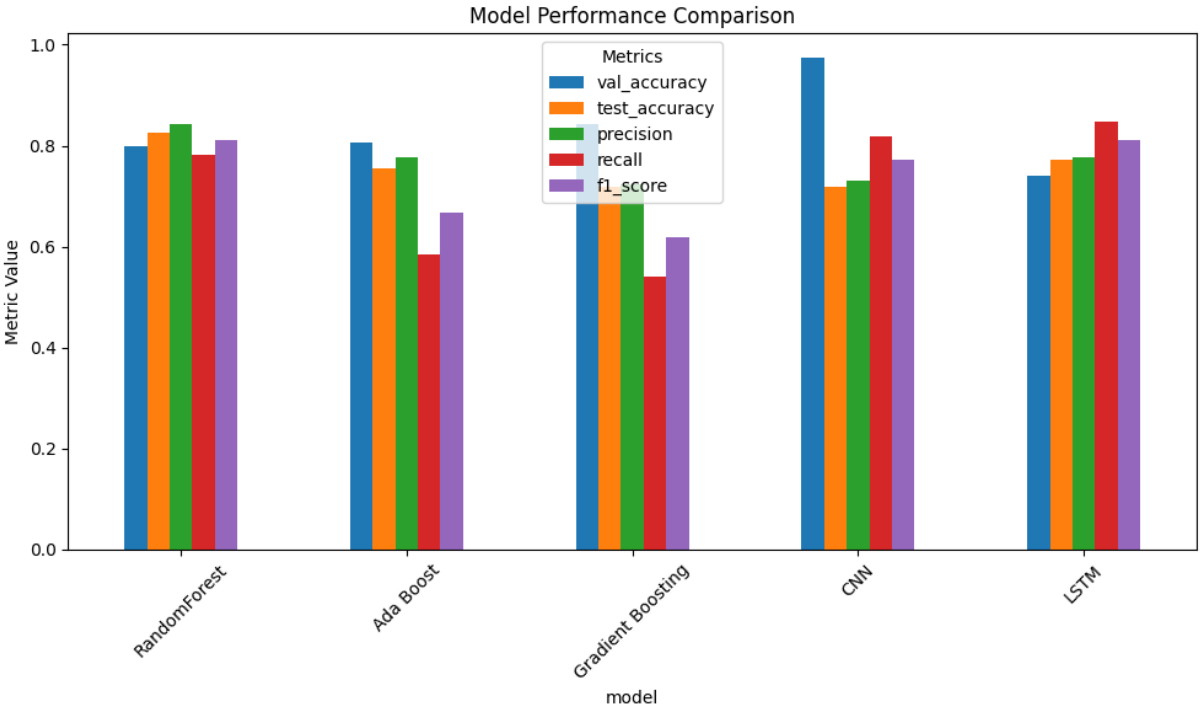

Figura 6.1: Representación visual de las métricas de rendimiento de los modelos.

6.1.2. Interpretación y análisis de los resultados

1. **Exactitud global (Accuracy):**

   - El modelo de **RandomForest** alcanza la mayor exactitud en el conjunto de prueba (82 %), seguido de **LSTM** (77 %). Esta tendencia confirma que ambos enfoques se ajustan correctamente a la distribución de datos.
   - Por su parte, el **CNN** y el **Gradient Boosting** obtienen una exactitud del 72 %. Cabe destacar que, en validación, el CNN llegó a mostrar una exactitud muy elevada (97 %), lo cual sugiere un posible sobreajuste: el modelo aprendió patrones de entrenamiento que no se generalizaron completamente al conjunto de prueba.
   - **Ada Boost** presenta una exactitud intermedia (75 %), situándose por debajo de RandomForest y LSTM, pero superando ligeramente a Gradient Boosting para esta métrica.

2. **Precisión (Precision):**

   - **RandomForest** exhibe la mayor precisión (84 %), es decir, de todas las predicciones de "gol" realizadas, un 84 % corresponden efectivamente a un evento real, indicando un bajo número de falsos positivos. Esto lo convierte en una opción robusta cuando se desea minimizar alertas erróneas.
   - Tanto **LSTM** como **Ada Boost** se sitúan en torno al 78 %. En el caso de LSTM, este valor combinado con su alto recall implica un equilibrio sólido entre detecciones verdaderas y falsas alarmas.
   - **Gradient Boosting** y **CNN** muestran valores de precisión ligeramente inferiores (72 % y 73 %, respectivamente), lo que revela que, aunque pueden identificar un gran número de eventos, también etiquetan incorrectamente algunos segmentos sin evento.

3. **Sensibilidad (Recall):**

   - El **LSTM** obtiene el mayor recall (85 %), lo cual refleja su capacidad para identificar la mayoría de los eventos reales presentes en el audio.
   - El **CNN** alcanza un recall del 82 %, demostrando también una buena detección de eventos reales, si bien su menor precisión sugiere que sacrifica algo de calidad en favor de cubrir un mayor número de eventos.
   - Por debajo, **RandomForest** registra un recall del 78 %, mientras que **Ada Boost** (58 %) y **Gradient Boosting** (54 %) obtienen resultados más modestos en términos de cobertura de eventos, lo que podría resultar insuficiente.

4. **F1-Score:**

   - El **F1-Score** combina precisión y recall, y en este caso **RandomForest** y **LSTM** empatan con un valor de 0.81, lo que indica que ambos modelos mantienen un equilibrio satisfactorio entre falsos positivos y falsos negativos.
   - El **CNN** alcanza un F1-score de 0.77, apoyado en su elevado recall; sin embargo, su menor precisión penaliza ligeramente la puntuación final en comparación con LSTM y RandomForest.
   - **Ada Boost** (0.67) y **Gradient Boosting** (0.62) quedan rezagados, reflejando tanto una menor cobertura de eventos como un mayor número de predicciones erróneas.

6.1.3. Detección del equipo

Además de entrenar los modelos para identificar la presencia de un gol en el audio del partido, se planteó la tarea adicional de determinar qué equipo era el responsable de la anotación (local o visitante). Para ello, todos los algoritmos (RandomForest, Ada Boost, Gradient Boosting, CNN y LSTM) se reentrenaron utilizando etiquetas que diferenciaban goles locales de goles visitantes. Sin embargo, debido a la escasez de muestras de goles visitantes en el conjunto de datos y a la limitada cantidad global de eventos de gol, los resultados obtenidos fueron insuficientes para la mayoría de los modelos.
En particular:

- La proporción desequilibrada de ejemplos provocó que RandomForest, Gradient Boosting, CNN y LSTM no lograran superar una precisión del 50 % en la clasificación del equipo anotador. Este bajo desempeño impide su uso fiable en un escenario real, donde la identificación del equipo es esencial.

- Solo el modelo Ada Boost alcanzó un rendimiento aceptable en esta tarea secundaria, consiguiendo una *precision* de 0.75 y un *recall* de 0.70 para goles visitantes, lo que le confirió un *F1-score* de 0.72. A pesar de no ser valores óptimos, estos resultados indican que Ada Boost tolera mejor el desbalance en la etiqueta de equipo.

6.1.4. Discusión de modelos escogidos para visualización

- El modelo **LSTM** destaca por su capacidad para capturar la mayor parte de los eventos reales (recall = 0.85), lo que resulta esencial en escenarios en los que perder jugadas relevantes podría afectar a la calidad final de un resumen automatizado. Aunque su precisión (0.78), el F1-Score (0.81) confirma que logra un buen compromiso entre sensibilidad y fiabilidad por ello fué escogido como uno de los modelos para detectar goles.

- El modelo **Ada Boost** muestra un rendimiento inferior en cobertura de eventos, lo cual podría resultar problemático. No obstante, debido a su rendimiento en la detección del

equipo al que se le atribuye el gol, se ha elegido también en la tarea de la detección de goles para visualización.

6.2. Visualización y validación manual de eventos detectados

Además del análisis cuantitativo realizado, se llevó a cabo una validación manual de los eventos detectados por el modelo para contrastar de forma cualitativa su desempeño y detectar posibles errores sistemáticos.
Se realizo la superposición visual de goles sobre la onda de audio. Se desarrollaron herramientas específicas que permitían observar las formas de onda de los partidos y, sobre ellas, marcar de forma clara los eventos detectados por el sistema.
A continuación, se muestran diferentes gráficas: la primera relativa a los goles reales sucedidos en uno de los partidos utilizados para verificar los resultados de los modelos (Figura 6.2), la segunda a la detección de los goles mediante el modelo LSTM (Figura 6.3) y por último, la gráfica que muestra la detección de goles por equipo de este mismo partido utilizando el modelo entrenado Ada Boost (Figura 6.4).

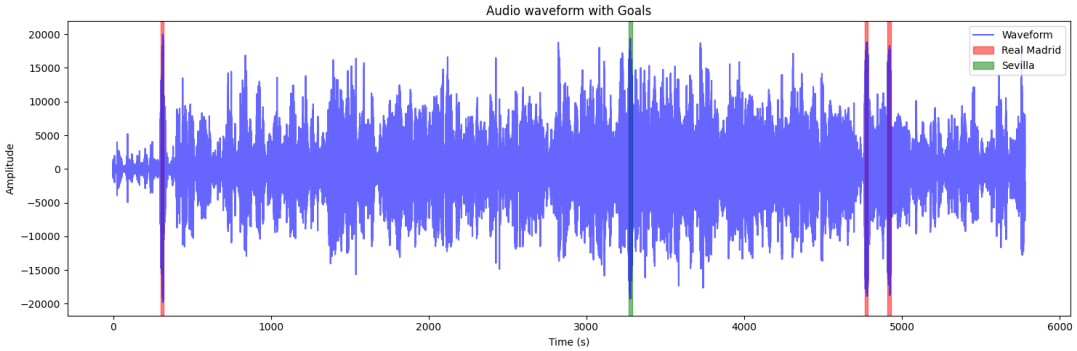

Figura 6.2: Distribución temporal de los goles reales en el partido de validación.

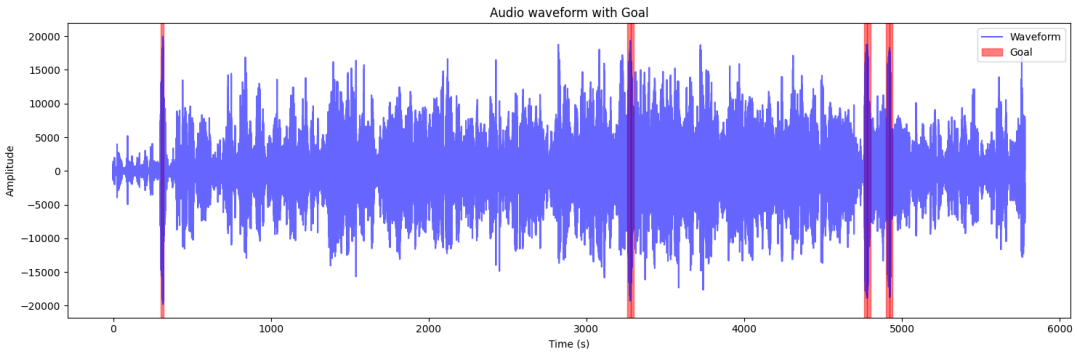

Figura 6.3: Detección de goles mediante el modelo LSTM en el mismo encuentro.

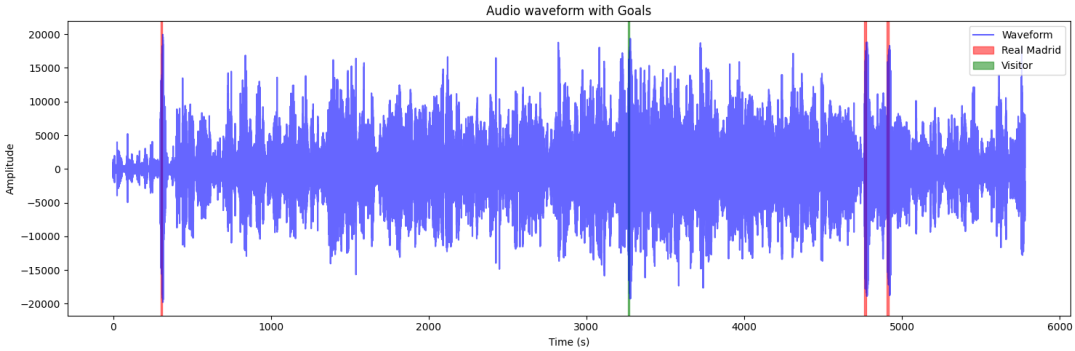

Figura 6.4: Detección de goles desglosada por equipo empleando el modelo Ada Boost.

Durante esta validación manual también se identificaron situaciones problemáticas que ayudaron a comprender las limitaciones actuales del modelo, como lo son sonidos del público muy intensos en momentos no relacionados con eventos relevantes y goles con baja intensidad sonora debido a la limpieza de comentarista del audio original

6.3. Discusión de los resultados

Una vez completadas las fases de entrenamiento, evaluación y validación manual del sistema, es posible realizar un análisis más global del desempeño del modelo y valorar en qué medida se han alcanzado los objetivos marcados al inicio del proyecto.

El principal objetivo de este trabajo fue el desarrollo de un sistema capaz de detectar goles en el audio ambiental de un partido de fútbol. Para ello, se establecieron metas específicas para las cuales a lo largo del proyecto se han desarrollado herramientas y metodologías que permiten cubrirlas. En particular, se ha logrado:

- Implementar un sistema robusto de preprocesado y limpieza del audio, mediante la creación de un programa específico de separación de fuentes.
- Extraer representaciones acústicas como el MFCC que resultan eficaces para el aprendizaje automático.
- Entrenar modelos de redes neuronales como LSTM, capaces de capturar patrones temporales relevantes en el audio.
- Validar tanto cuantitativamente (precisión, recall, F1-score) como cualitativamente (superposición visual) la calidad de las predicciones.

Los resultados obtenidos muestran que el sistema es eficaz y ha demostrado que incluso en un entorno tan ruidoso y variable como un estadio de fútbol, el uso adecuado de técnicas de procesamiento de audio y aprendizaje automático puede ofrecer resultados prometedores. Sin embargo, a lo largo del proyecto también se han identificado algunas limitaciones, como:

- La falta de un conjunto de datos más amplia y anotada con precisión, que impide alcanzar resultados de tipo industrial, aunque los obtenidos son útiles para investigación.
- La dependencia del tipo de grabación, partidos con mucho ruido de fondo o problemas de sincronización complicaban el análisis.
- El coste computacional del procesamiento de audios largos y detallados, que obligó a ajustar parámetros y frecuencias de muestreo.

- La mala calidad del audio inicial, sumado a la necesidad de eliminación del comentarista y la variabilidad entre las distintas grabaciones del adudio, incluso entre las distintas partes de un mismo evento.

Pese a estas dificultades, el sistema mostró algunas fortalezas notables:

- Capacidad para detectar patrones temporales característicos, especialmente tras la incorporación de redes LSTM, que permitieron modelar mejor la dinámica temporal del audio.
- Resistencia al ruido tras la aplicación de técnicas de normalización y limpieza, lo que mejoró la claridad de los datos de entrada y la precisión en las predicciones.
- Potencial para ser escalado a más clases de eventos, al haber establecido una base sólida para la detección basada en características acústicas.

En conjunto, el sistema desarrollado cumple con los objetivos propuestos y ofrece una base sólida para seguir explorando la detección automática de eventos sonoros en entornos complejos. La combinación de técnicas de aprendizaje automático con herramientas de preprocesado y validación visual ha resultado efectiva y demuestra el potencial del análisis de audio como complemento al análisis tradicional de datos deportivos.

7 Líneas de trabajo futuras

Aunque los resultados obtenidos en este trabajo son satisfactorios y permiten validar la viabilidad del enfoque propuesto, existen numerosas oportunidades de mejora y expansión del sistema que podrían abordarse en trabajos futuros. A continuación, se describen algunas de las líneas más relevantes:

- **Ampliación y mejora del conjunto de datos**
  Una de las principales limitaciones del sistema actual ha sido el número y la calidad de los audios disponibles. Contar con un mayor volumen de partidos, idealmente sin comentarista y con buena calidad de grabación, permitiría entrenar modelos más robustos, detectar más tipos de eventos y mejorar la generalización.

- **Mejora de la precisión temporal**
  En el sistema actual, la alineación entre los eventos anotados y el audio no siempre es precisa al segundo, lo que puede afectar a la calidad del entrenamiento y a la evaluación del modelo. Desarrollar métodos automáticos de ajuste temporal basados en picos de sonido, cambios de energía o sincronización con otros datos podría aumentar significativamente la precisión y coherencia de los resultados.

- **Implementación de un sistema en tiempo real**
  Aunque el sistema actual se ejecuta de forma offline, una línea futura interesante sería el desarrollo de una versión en tiempo real que permita detectar eventos mientras se emite el partido.

- **Incorporación del análisis de sentimientos**
  Una línea de trabajo especialmente prometedora es la incorporación del análisis de sentimientos a las señales acústicas registradas en el estadio. Este enfoque no solo complementa la detección de eventos, sino que añade una capa de interpretación emocional sobre el contexto del partido, lo que puede resultar útil en ámbitos como la retransmisión deportiva, el estudio del comportamiento de la afición o la analítica avanzada aplicada al deporte.

- **Extensión a otros eventos y deportes**
  Hasta ahora, el sistema ha sido diseñado y entrenado con audios de partidos de fútbol para detectar goles. Una línea de trabajo futura consiste en adaptar y validar el modelo en diferentes eventos (faltas, tiros libres...) y disciplinas deportivas. Esta extensión implicaría:

  - Recopilar y etiquetar audios específicos de cada deporte, identificando sus eventos característicos.
  - Evaluar la capacidad de generalización del sistema y, en caso necesario, explorar estrategias de transferencia de aprendizaje.
  - Definir métricas comunes y específicas para comparar resultados entre deportes y garantizar una evaluación homogénea.

8 Conclusiones

Este Trabajo de Fin de Grado ha abordado el desarrollo de un sistema para la detección de eventos sonoros significativos en partidos de fútbol, a partir del análisis del audio ambiental. Esta línea de trabajo, en la intersección entre el procesamiento digital de señales y el aprendizaje automático, supone una aportación novedosa dentro del campo de la analítica deportiva, que tradicionalmente ha priorizado el análisis de datos estructurados frente a fuentes menos exploradas como el sonido.
A lo largo del desarrollo del proyecto, se han conseguido cumplir los objetivos propuestos inicialmente, estructurando el trabajo en torno a varias fases fundamentales:

- Análisis exploratorio de los datos disponibles, que permitió comprender las características del audio real de los partidos, sus limitaciones, la presencia de ruido y las diferencias entre partidos con y sin comentarista.
- Diseño y desarrollo de un conjunto de herramientas personalizadas para el procesamiento de audio, que incluyó la limpieza del sonido, la segmentación temporal, la extracción de características acústicas relevantes como los coeficientes MFCC, así como la normalización e integración con los eventos registrados.
- Entrenamiento de modelos de aprendizaje automático, que mostraron ser capaces de aprender patrones acústicos temporales asociados a goles, ofreciendo resultados prometedores incluso en escenarios ruidosos o con fuentes de audio complejas.
- Validación de los modelos, tanto desde un punto de vista cuantitativo como cualitativo, mediante la comparación con los eventos reales del partido y el uso de visualizaciones superpuestas al audio que facilitaron la interpretación de los resultados.

Un aspecto especialmente valioso del proyecto ha sido el uso disciplinado de metodologías ágiles, combinadas con herramientas como GitLab, que han permitido organizar el trabajo en sprints y milestones, mantener un control claro del progreso a través de issues y registrar el conocimiento generado en una Wiki.

En conjunto, el sistema desarrollado representa un primer paso efectivo hacia la automatización del análisis del sonido en contextos deportivos. Si bien el sistema es aún preliminar y existen márgenes amplios de mejora, ha demostrado ser funcional, extensible y con potencial para escalarse.

En particular, concluimos que es posible realizar análisis de sentimiento a partir del audio de un acontecimiento deportivo, dado que las variaciones en la intensidad del sonido, los picos de ruido generados por reacciones masivas de la audiencia y la modulación en el timbre de los cánticos reflejan estados emocionales concretos. Además, el análisis temporal de estos picos sonoros permitiría caracterizar la evolución emocional del público a lo largo del partido.

Bibliografía

[1] R. Das and T. D. Singh, "Multimodal sentiment analysis: A survey of methods, trends, and challenges," *ACM Computing Surveys (CSUR)*, vol. 55, no. 12, pp. 1–39, 2023.

[2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5 – 32, 2001.

[3] H. Phan, M. Maass, R. Mazur, and A. Mertins, "Acoustic event detection and localization with regression forests," in *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2014. [Online]. Available: https://kar.kent.ac.uk/72695/

[4] S. Misra and H. Li, "Chapter 9 - Noninvasive fracture characterization based on the classification of sonic wave travel times," in *Machine Learning for Subsurface Characterization*, S. Misra, H. Li, and J. He, Eds. Gulf Professional Publishing, 2020, pp. 243–287. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128177365000090

[5] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, 1997.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] C. Olah, "Understanding LSTM networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015, accessed: 2025-04-08.

[8] M. Galar Idoate, "Visualizando neuronas en redes neuronales convolucionales," 2020, accedido: 2025-04-08. [Online]. Available: https://academica-e.unavarra.es/handle/2454/33694

[9] M. Mehta, "A review on sentiment analysis of text, image and audio data," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 7, no. 3, pp. 245–251, 2021. [Online]. Available: https://www.researchgate.net/publication/352832072_A_Review_on_Sentiment_Analysis_of_Text_Image_and_Audio_Data

[10] D. Stoller, S. Ewert, and S. Dixon, "Wave-u-net: A multi-scale neural network for end-to-end audio source separation," in *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, Paris, France, September 2018, pp. 334–340. [Online]. Available: https://arxiv.org/abs/1806.03185

[11] S. Rouard, F. Massa, and A. Défossez, "Hybrid transformers for music source separation," 2022. [Online]. Available: https://arxiv.org/abs/2211.08553

[12] R. Bhagat and R. Kaur, "Improved audio filtering using extended high pass filters," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 7, 2013.

[13] M. A. Salih, "Audio noise reduction using low pass filters," *Open Access Library Journal*, vol. 4, 2017. [Online]. Available: https://www.scirp.org/journal/paperinformation?paperid=80233

[14] A. M. El-Samie, "Noise removal in speech processing using spectral subtraction," *Circuits and Systems*, vol. 5, 2014. [Online]. Available: https://www.scirp.org/journal/paperinformation?paperid=45989

[15] G. Poch Lacalle, "Normalización del volumen de archivos de sonido en formato mp3," Barcelona, Spain, 2019, disponible en el repositorio institucional O2 de la UOC. [Online]. Available: https://openaccess.uoc.edu/bitstream/10609/716/1/26980tfc.pdf

[16] K. Schwaber and J. Sutherland, "The Scrum guide," Disponible en línea: https://scrumguides.org/scrum-guide.html (consultado el 1 de julio de 2025), 2020.

[17] D. J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*. Sequim, WA, USA: Blue Hole Press, 2010.

[18] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

A Enlace a la documentación del código

La documentación generada del código se encuentra disponible en el siguiente enlace:
https://hmb899.github.io/unican-listentothestadium/src.html

B README del proyecto

UNICAN-ListenToTheStadium

Description

UNICAN-ListenToTheStadium is a project aimed at analyzing the sentiment of attendees at a football match based on the relationship between the ambient sound of the game and the events happening on the field, with potential application to other sports.

At the moment, the project is focused on analyzing the audio environment of football matches to detect and classify goals based on the crowd's acoustic response. The system includes preprocessing of match data, training of machine learning models, and goal detection, with possible applications in other sports as well.

Project Structure

The project is organized into the following main folders and files:

- **Makefile**: Manages execution of the entire pipeline. Running `make run` will process data, train models, and detect goals for all matches. If a match number is specified (e.g., `make run MATCH=2`), it will process and detect goals for that specific match using pre-trained models.
- **Src/**: Contains the core Python scripts:
  - `__init__.py`
  - `constants.py`: Contains the common constants of the programs.
  - `utilities.py`: Contains the common functions shared between programs.
  - `match_mapping.py`: Maps match numbers to their respective abbreviations.
  - `data_preparation/`:
    - `Json_Data.py`: Generates structured CSVs.
    - `audio_cleaner.py`: Removes the commentator's voice from audio, filters the audio and normalizes it.
    - `audio_analysis.py`: Generates graphs for the analysis of the match audio.
    - `search_goals.py`: Searches for potential goal-related events.
  - `goals_classification/`:
    - `goals_success.py`: Determines success status of identified goals.
    - `split_goals.py`: Separates goal audio segments.
  - `training/`:
    - `random_forest.py`, `ada_boost.py`, `gradient_boosting.py`: Traditional ML models.
    - `cnn.py`, `cnn_1d.py`, `lstm.py`: Deep learning models.

- `goal_classification/`:
  - `model_comparator.py`: Compares performance of trained models.
  - `goal_detector_ada_boost.py`, `goal_detector_lstm_model.py`: Goal detection using trained models.

- **Data/**:
  - `Match_tables/`: Match data organized per match (e.g., `Match_RMA-LPA`) containing videos, event JSONs, and generated files. Now also includes:
    - Each subfolder follows the naming convention `Match_{match_abbreviations}` (e.g., `Match_RMA-LPA`).
    - Inside each subfolder are:
      - **Match video** (in `.mp4` format, named with the match abbreviation).
      - **Match data** in a JSON file ({`abbreviation`}`_all_events.json`).
      - **Files generated** by the programs, such as processed CSV files and extracted audios.
      - `Audio/`: Audio segments by match.
      - `Graphs/`: Visual representations (amplitude, frequency, dB, etc.).
  - `Audio_goals/`:
    - `goals/`: Audio clips labeled as actual goals.
    - `no_goals/`: Audio clips not corresponding to real goals.
  - `Models/`: Contains all trained ML and DL models.
  - `Processed/`: Contains all generated CSVs, including detailed data per goal.

- **Scripts/**:
  - `data_process`: Runs all preprocessing and data transformation scripts in `Code`.
  - `detect_goals`: Applies trained models to detect goals in matches.
  - `goal_classification`: Organizes detected audio clips into `goals` and `no_goals`.
  - `train_models`: Trains machine learning and deep learning models.