

Pontificia Universidad Javeriana Cali  
Facultad de Ingeniería y Ciencias.  
Ingeniería de Sistemas y Computación.  
Proyecto de Grado.

# Identificación automática de patologías en la voz a partir de técnicas de aprendizaje automático

Carlos Felipe Palacio Lozano  
Juan Esteban Acosta López

Director: Dr. Julián Gil González

15/08/2024



Santiago de Cali, 15/08/2024.

Señores

**Pontificia Universidad Javeriana Cali.**

Dr. Gerardo Mauricio Sarria Montemiranda

Director Carrera de Ingeniería de Sistemas y Computación.

Cali.

Cordial Saludo.

Por medio de la presente me permito informarle que los estudiantes de Ingeniería de Sistemas y Computación Carlos Felipe Palacio Lozano (cod: 8956397) y Juan Esteban Acosta López(cod: 8952309) trabajaron bajo mi dirección en el proyecto de grado titulado “Identificación automática de patologías en la voz a partir de técnicas de aprendizaje automático” el cual se encuentra finalizado y listo para sustentación.

Atentamente,

A handwritten signature in black ink, reading 'Julián Gil González', written in a cursive style.

---

Dr. Julián Gil González

Santiago de Cali, 15/08/2024.

Señores

**Pontificia Universidad Javeriana Cali.**

Dr. Gerardo Mauricio Sarria Montemiranda

Director Carrera de Ingeniería de Sistemas y Computación.

Cali.

Cordial Saludo.

Nos permitimos presentar a su consideración el trabajo de grado de grado titulado “Identificación automática de patologías en la voz a partir de técnicas de aprendizaje automático” con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el proyecto de grado y posteriormente optar al título de Ingeniero de Sistemas y Computación.


Al firmar aquí, damos fe que entendemos y conocemos las directrices para la presentación de trabajos de grado de la Facultad de Ingeniería y Ciencias aprobadas el 26 de Noviembre de 2009, donde se establecen los plazos y normas para el desarrollo del anteproyecto y del trabajo de grado.

Atentamente,



Carlos Felipe Palacio Lozano

Código: 8956397



Juan Esteban Acosta López

Código: 8952309

# Resumen

El proyecto de investigación profundizó en el desarrollo de múltiples modelos de aprendizaje automático, estableciendo una comparativa entre las técnicas clásicas de aprendizaje y las cada vez más emergentes técnicas de aprendizaje profundo, llevando a cabo una tarea de clasificación en el ámbito de la salud. El problema central de esta investigación consistió en determinar de manera precisa si un modelo de aprendizaje automático podía identificar la presencia de patologías en la voz. El enfoque adoptado para ambas técnicas de aprendizaje fue diferenciado, permitiendo a los modelos de cada técnica centrarse en distintas aproximaciones para resolver el mismo problema.

Varias tareas realizadas previamente al desarrollo de los modelos contribuyeron a mejorar sus resultados, demostrando cómo, mediante ciertas estrategias, se pueden superar limitaciones como la cantidad de datos disponibles, logrando así un mejor desempeño. Finalmente, al completar los experimentos, se desarrolló una interfaz gráfica que permite interactuar con dos de los modelos creados, destacando los mejores resultados obtenidos en cada una de las técnicas de aprendizaje seleccionadas.

Los resultados finales resaltan cómo el aprendizaje automático desempeña un papel diferencial en tareas complejas para los seres humanos, ya sea por su naturaleza imperceptible o por la necesidad de analizar grandes volúmenes de datos para generar resultados en tiempo real. En conclusión, esta investigación evidencia el potencial del aprendizaje automático en la detección de patologías en la voz, logrando resultados precisos a pesar de contar con una cantidad limitada de datos en comparación con otros problemas. Además, abre el camino para futuros refinamientos y estudios clínicos que incluyan los modelos en entornos reales, permitiendo obtener una retroalimentación no observable en las etapas de desarrollo.

**Palabras Clave:** Aprendizaje profundo, aprendizaje clásico, red neuronal, patologías, voz.

# Índice general

<b>1. Análisis</b>	<b>13</b>
1.1. Planteamiento del Problema . . . . .	13
1.1.1. Formulación . . . . .	14
1.1.2. Sistematización . . . . .	14
1.2. Objetivos . . . . .	14
1.2.1. Objetivo General . . . . .	14
1.2.2. Objetivos Específicos . . . . .	14
1.3. Justificación . . . . .	15
1.4. Delimitaciones y Alcances . . . . .	16
1.4.1. Delimitaciones . . . . .	16
1.4.2. Alcance . . . . .	16
1.4.3. Entregables . . . . .	16
1.5. Resultados Esperados . . . . .	16
1.6. Metodología . . . . .	17
1.6.1. Tipo de Estudio . . . . .	17
1.6.2. Actividades . . . . .	17
1.6.3. Objetivo 1: Desarrollar un módulo de preprocesamiento de señales de la voz que limpie y verifique la correctitud de los datos . . . . .	17
1.6.4. Objetivo 2: Desarrollar modelos de aprendizaje automático para la clasifica- ción binaria de señales de voz . . . . .	17
1.6.5. Objetivo 3: Realizar una búsqueda de hiperparámetros para los modelos pro- puestos . . . . .	17
1.6.6. Objetivo 4: Desarrollar un módulo de evaluación para validar los modelos de aprendizaje automático utilizando métricas estándar . . . . .	18
1.6.7. Objetivo 5: Desarrollar una aplicación web que permita desplegar los resultados del modelo . . . . .	18
<b>2. Marco de referencia</b>	<b>19</b>
2.1. Áreas Temáticas . . . . .	19
2.2. Marco Teórico . . . . .	20
2.2.1. Tipos de Aprendizaje . . . . .	20
2.2.2. Aprendizaje Supervisado . . . . .	20
2.2.3. Aprendizaje profundo . . . . .	24
2.2.4. Comparativa entre modelos . . . . .	28
2.2.5. Evaluación y Validación . . . . .	30
2.2.6. Lectura de muestras de audio. . . . .	31
2.2.7. Patologías en la voz. . . . .	32

2.3. Trabajos Relacionados . . . . .	33
2.3.1. Automatic Assessment of Voice Quality in the Context of Multiple Annotations [1] . . . . .	33
2.3.2. A review on voice pathology: Taxonomy, diagnosis, medical procedures and detection techniques, open challenges, limitations, and recommendations for future directions [2] . . . . .	33
2.3.3. Identification of Pathological Voices Using Glottal Noise Measures [3] . . . . .	33
<b>3. Conjunto de Datos . . . . .</b>	<b>35</b>
3.1. Recolección de datos. . . . .	35
3.1.1. Especificación de los datos. . . . .	35
3.2. Lectura de las muestras. . . . .	36
3.3. Preprocesamiento técnicas de aprendizaje clásicas. . . . .	37
3.4. Preprocesamiento técnicas de aprendizaje aprendizaje profundo . . . . .	38
3.4.1. Aumento de datos técnicas clásicas. . . . .	39
3.4.2. Aumento de datos técnicas de aprendizaje profundo . . . . .	41
3.4.3. Conjuntos de entrenamiento y prueba. . . . .	41
3.4.4. Visualización de las muestras. . . . .	42
<b>4. Desarrollo e implementación . . . . .</b>	<b>45</b>
4.1. Flujo de datos. . . . .	45
4.2. Técnicas clásicas de machine learning. . . . .	46
4.2.1. Entrenamiento . . . . .	46
4.2.2. Evaluación de los modelos . . . . .	47
4.2.3. Optimización . . . . .	47
4.2.4. Implementación Regresión logística . . . . .	47
4.2.5. Implementación K-Nearest Neighbors (KNN) . . . . .	48
4.2.6. Implementación árboles de decisión . . . . .	48
4.2.7. Implementación Random Forest . . . . .	48
4.3. Técnicas de aprendizaje profundo . . . . .	49
4.3.1. Red Neuronal Convolucional (CNN) . . . . .	49
4.3.2. Entrenamiento . . . . .	50
<b>5. Resultados . . . . .</b>	<b>55</b>
5.1. Regresión logística . . . . .	55
5.1.1. Modelo base . . . . .	55
5.1.2. Modelo optimizado . . . . .	56
5.2. KNN . . . . .	57
5.2.1. Modelo base . . . . .	57
5.2.2. Modelo optimizado . . . . .	58
5.3. Árboles de decisión . . . . .	59
5.3.1. Modelo base . . . . .	59

---

5.3.2. Modelo optimizado . . . . .	60
5.4. Random forest . . . . .	61
5.4.1. Modelo base . . . . .	61
5.4.2. Modelo optimizado . . . . .	62
5.5. Red Neuronal Convulucional . . . . .	63
5.5.1. Modelo base . . . . .	63
5.5.2. Modelo optimizado . . . . .	64
5.6. Comparativa de modelos . . . . .	66
5.6.1. Mejor modelo . . . . .	67
<b>6. Despliegue</b>	<b>69</b>
6.1. Despliegue de la Aplicación de Detección de Voz Patológica . . . . .	69
6.1.1. Estructura del Proyecto . . . . .	69
6.1.2. Carga y Uso de Modelos de Machine Learning . . . . .	69
6.1.3. Interfaz de Usuario y Estilización . . . . .	70
6.1.4. Procesamiento de Archivos de Audio . . . . .	70
6.1.5. Despliegue y Funcionamiento . . . . .	70
<b>7. Conclusiones</b>	<b>73</b>
7.1. Conclusiones . . . . .	73
7.2. Trabajo futuro . . . . .	74
7.2.1. Identificación del tipo de patología . . . . .	74
7.2.2. Uso de aprendizaje semisupervisado . . . . .	74
<b>Bibliografía</b>	<b>75</b>

# Introducción

Las patologías de la voz representan una preocupación significativa en el ámbito de la salud, afectando tanto a la comunicación diaria como al desempeño profesional de muchas personas. La voz es una herramienta esencial no solo para la interacción social, sino también para profesiones que dependen de ella, como la docencia, el canto, la locución y las conferencias. Sin embargo, diversas condiciones médicas pueden comprometer la calidad vocal, desde la laringitis hasta los nódulos vocales y los carcinomas, cada una con implicaciones para la salud y el bienestar de los afectados.

La detección temprana y precisa de las patologías de la voz es crucial para mitigar sus efectos y facilitar un tratamiento efectivo. Sin embargo, este proceso enfrenta varios desafíos, incluyendo la falta de síntomas claros en las etapas iniciales y la reticencia de algunas personas a buscar atención médica. Además, los métodos de diagnóstico actuales, que incluyen evaluaciones clínicas y el uso de dispositivos como laringoscopios y videostroboscopios, aunque efectivos, son a menudo invasivos, costosos y requieren equipos especializados que no siempre están disponibles en todas las instituciones de salud y que aumentan la brecha social en cuanto al acceso a la medicina en países en vías de desarrollo o subdesarrollados.

La creciente incidencia de trastornos vocales subraya la necesidad de desarrollar métodos de diagnóstico más accesibles y menos invasivos. En este contexto, la inteligencia artificial y, específicamente, el aprendizaje automático, ofrecen nuevas oportunidades para mejorar la detección y el manejo de las patologías de la voz. Estas tecnologías permiten el análisis detallado de señales vocales, identificando patrones y anomalías que pueden no ser perceptibles mediante métodos tradicionales. Al aprovechar grandes volúmenes de datos y algoritmos avanzados, los modelos de aprendizaje automático pueden proporcionar diagnósticos precisos y oportunos, reduciendo así la necesidad de procedimientos invasivos y mejorando la accesibilidad del diagnóstico.

La integración de modelos de aprendizaje automático en el diagnóstico de patologías vocales promete transformar la práctica clínica, permitiendo una detección más temprana y precisa de estas afecciones. Esto no solo mejoraría la precisión diagnóstica, sino también ofrecería una herramienta práctica y eficiente que pueda ser utilizada tanto por especialistas como por médicos de atención primaria.

En las siguientes secciones, exploraremos el papel que pueden tener los modelos de aprendizaje automático en la identificación de patologías vocales. Se analizarán las técnicas utilizadas para procesar y clasificar señales vocales, así como los beneficios y desafíos asociados con la implementación de estas tecnologías en el ámbito clínico.



## 1.1. Planteamiento del Problema

Las patologías de la voz tales como laringitis, nódulos, parálisis, carcinomas, entre otros, generalmente son detectados de forma tardía. Específicamente, según la Sociedad Española de Otorrinolaringología y Cirugía de Cabeza y Cuello entre un 8 a 10 % de la población mundial padece un trastorno en la voz [4]. Como es evidente, muchos de estos casos no son detectados a tiempo, generando un agravamiento sustancial del trastorno. Es importante mencionar que algunas personas se muestran renuentes ante el acudimiento a un profesional sanitario, lo que complica el diagnóstico y tratamiento, ralentizando la detección de la patología. Por lo tanto, las personas que dependen de su voz tales como educadores, conferencistas, cantantes, entre otros, tienen repercusiones en su vida personal y laboral, teniendo consecuencias en el estado psicológico de los afectados [5], sin mencionar las posibles consecuencias físicas del agravamiento de las enfermedades.

Las enfermedades de la voz pueden afectar significativamente a las personas que dependen de ella para su vida diaria o profesional. Algunos trastornos comunes incluyen la laringitis, que es la inflamación de las cuerdas vocales causando voz ronca o pérdida de la voz; parálisis o paresia de las cuerdas vocales, que puede ser causada por infecciones virales, cirugías, accidentes cerebrovasculares o cáncer, afectando la respiración o la calidad de la voz; y la disfonía espasmódica, un problema nervioso que causa espasmos en las cuerdas vocales, afectando la calidad de la voz [6]. Estos trastornos pueden ser causados por una variedad de factores incluyendo crecimientos anormales, inflamación, problemas nerviosos, desequilibrios hormonales o abuso vocal.

El diagnóstico desde el ámbito médico de las enfermedades de la voz, generalmente comienza con una evaluación clínica detallada, la cuál incluye un examen físico y análisis de los síntomas. Posteriormente, al sospechar de una posible patología, se utilizan instrumentos como laringoscopios y videostroboscopios, los cuales permiten visualizar las cuerdas vocales y evaluar su movimiento y estado [7]. Sin embargo, estas herramientas no permiten al médico conocer detalles específicos como la naturaleza celular o histológica de las lesiones, lo que es fundamental para el diagnóstico definitivo. A pesar de ser una herramienta diagnóstica valiosa, las biopsias conllevan riesgos como infección o sangrado y pueden resultar invasivas, lo que añade una capa de estrés y complejidad tanto para el paciente como para el médico a cargo del procedimiento. Las patologías en la voz pueden ser difíciles de diagnosticar en etapas tempranas para un profesional de la salud de primera instancia o médicos de cabecera, debido a la falta de especialización del mismo y los síntomas inespecíficos del paciente pueden apuntar a otra enfermedad, además de la carencia de equipos especializados como

en endoscopios, videostroboscopios o laringoscopios por parte de la institución de salud [8].

La aplicación del aprendizaje automático en el diagnóstico de patologías vocales representa un avance significativo, permitiendo el análisis exhaustivo de señales de voz de manera no invasiva. Al entrenar modelos con extensos conjuntos de datos, pueden detectar patrones y anomalías que a menudo son imperceptibles para los métodos tradicionales. Esto no solo mejora la precisión del diagnóstico sino que también minimiza la incomodidad y los riesgos asociados con procedimientos más invasivos como las biopsias.

### 1.1.1. Formulación

¿Cómo implementar un sistema de identificación de patologías en la voz a partir de análisis de señales de voz utilizando modelos de aprendizaje automático?

### 1.1.2. Sistematización

1. Preprocesamiento. (¿Cómo desarrollar un módulo de preprocesamiento de las señales de la voz?)
2. Selección de modelos. (¿Cómo desarrollar modelos de aprendizaje automático para identificar patologías en la voz?)
3. Optimización (búsqueda de hiperparámetros). (¿Cómo realizar una búsqueda de hiperparámetros para los modelos planteados?)
4. Evaluación. (¿Cómo evaluar los modelos utilizando las métricas de problemas de clasificación?)
5. Despliegue. (¿Cómo desplegar los resultados del modelo mediante una aplicación web?)

## 1.2. Objetivos

### 1.2.1. Objetivo General

Implementar un modelo de clasificación de señales a partir de técnicas de aprendizaje automático para la identificación de patologías en la voz.

### 1.2.2. Objetivos Específicos

1. Desarrollar un módulo de preprocesamiento de señales de la voz que limpie y verifique la correctitud de los datos.
2. Desarrollar modelos de aprendizaje automático para la clasificación binaria de señales de voz.
3. Realizar una búsqueda de hiperparámetros para los modelos propuestos.

4. Desarrollar un módulo de evaluación para validar los modelos de aprendizaje automático utilizando métricas estándar.
5. Desarrollar una aplicación web que permita desplegar los resultados del modelo.

### 1.3. Justificación

El modelo de aprendizaje automático propuesto podría permitir reducir los tiempos en la detección de patologías en la voz frente a los métodos tradicionales. Tradicionalmente, el diagnóstico de enfermedades en la voz ha sido un desafío para los médicos de cabecera, por lo tanto, ayudar a los profesionales de la salud en el diagnóstico de patologías de la voz causadas por el desgaste y mal uso de esta, permitiendo remitir al paciente de forma asertiva y temprana donde un otorrinolaringólogo o algún especialista médico en el tema, suponiendo así una disminución en los costos y evita tener un tratamiento invasivo que requiera intervención quirúrgica, permitiendo al paciente retomar sus actividades laborales y sociales de forma más rápida y efectiva.

El proyecto se enfoca en asistir a médicos de atención primaria en la detección y evaluación de la calidad de voz de los pacientes. Su objetivo es minimizar los diagnósticos erróneos en servicios de salud, agilizando la remisión de pacientes con posibles patologías vocales a especialistas o tratamientos. Esto es crucial para una rápida identificación y tratamiento de trastornos de la voz, evitando el empeoramiento debido a diagnósticos iniciales incorrectos. Resulta particularmente beneficioso para individuos cuyas profesiones o actividades diarias dependen intensamente de su voz, como educadores, locutores, guías turísticos, presentadores, periodistas, políticos y cantantes, al proporcionarles un diagnóstico temprano y preciso.

A su vez, se han realizado trabajos similares con esta base de datos y entorno al tema. En consecuencia, existe una gran literatura y bases sólidas planteadas por otros autores en el tema. Tal como presenta Hedge et al. [9] recopilan 98 trabajos en el ámbito del machine learning y el diagnóstico de enfermedades en la voz, también desglosan el uso de los algoritmos más comunes de machine learning presentes en cada trabajo como SVM, ANN, KNN, K-Medias y clasificadores lineales, obteniendo en promedio mas de 90 % de accuracy. Como también, Alhussein y Muhammad [10] exponen el uso del deep learning en la identificación de las patologías de la voz, usando CNN y obteniendo un 97.5 % de accuracy.

Otro elemento a tener en cuenta es el uso de herramientas gratuitas como Google Colab o Kaggle que proporcionan una gran capacidad de cómputo, además de la participación del director Julián Gil Gonzalez que cuenta con una amplia experiencia y conocimiento en el ámbito del aprendizaje automático y que previamente ha trabajado en proyectos relacionados a patologías en la voz, también desde el ámbito del machine learning.

## 1.4. Delimitaciones y Alcances

### 1.4.1. Delimitaciones

- Dataset con 226 señales de voz.
- Calidad de las señales de voz.
- No habrá validación clínica de los resultados.

### 1.4.2. Alcance

- Modelo de aprendizaje automático que identifique patologías en la voz.
- La clasificación multiclase de las patologías presentes en el dataset y la validación clínica de los resultados será objeto de estudio de futuros trabajos.

### 1.4.3. Entregables

- Modelo de aprendizaje automático.
- Aplicación web que despliegue los resultados.
- Documento de tesis.

## 1.5. Resultados Esperados

- **Modelo de aprendizaje automático funcional:** Se espera desarrollar un modelo de aprendizaje automático funcional y preciso que pueda identificar patologías de la voz con una alta tasa de precisión en una variedad de casos clínicos.
- **Evaluación rigurosa:** Se realizará una evaluación rigurosa del modelo de aprendizaje automático mediante métricas de rendimiento estándar, lo que incluye precisión, recall, F1-score y otros indicadores relevantes.
- **Documentación integral:** Se entregará una documentación completa y clara de todo el proceso, lo que facilitará la comprensión y replicación del proyecto.
- **Comunicación de resultados:** Los resultados y hallazgos del proyecto se comunicarán de manera efectiva a través de un documento de tesis y una aplicación web que despliegue los resultados del modelo.

## 1.6. Metodología

### 1.6.1. Tipo de Estudio

El proyecto es un estudio experimental cuantitativo. La naturaleza experimental del proyecto radica en la creación y validación de un modelo de aprendizaje profundo que identifique patologías en la voz sirviendo como insumo y apoyo para los médicos de cabecera en la detección de las enfermedades.

Para el desarrollo y validación del proyecto se utilizará la metodología CRISP-DM. Esta metodología es comúnmente utilizada en proyectos de aprendizaje automático y minería de datos, proporcionando un enfoque estructurado para el proyecto y un sistema óptimo para el cumplimiento de los objetivos del mismo.

### 1.6.2. Actividades

#### 1.6.3. Objetivo 1: Desarrollar un módulo de preprocesamiento de señales de la voz que limpie y verifique la correctitud de los datos

1. Investigar algoritmos de limpieza de señales de voz.
2. Implementar algoritmos para la limpieza y verificación de los datos de la voz.
3. Realizar técnicas de data augmentation dada la cantidad limitada de datos, con el fin de obtener mejores resultados en las etapas posteriores.

#### 1.6.4. Objetivo 2: Desarrollar modelos de aprendizaje automático para la clasificación binaria de señales de voz

1. Investigar modelos de aprendizaje automático específicas para la clasificación de señales de voz.
2. Implementar el diseño de los modelos seleccionados.
3. Entrenar los modelo con un conjunto de datos etiquetado.
4. Realizar pruebas preliminares del modelo para evaluar su rendimiento y funcionamiento.

#### 1.6.5. Objetivo 3: Realizar una búsqueda de hiperparámetros para los modelos propuestos

1. Identificar los hiperparámetros clave que afectan el rendimiento del modelo.
2. Implementar estrategias para la búsqueda de hiperparámetros, como la búsqueda en malla o búsqueda aleatoria.
3. Realizar la búsqueda y selección de los mejores hiperparámetros.
4. Reentrenar el modelo con los hiperparámetros optimizados y evaluar su rendimiento.

**1.6.6. Objetivo 4: Desarrollar un módulo de evaluación para validar los modelos de aprendizaje automático utilizando métricas estándar**

1. Investigar métricas estándar de evaluación para modelos de clasificación, como precisión, recall y F1-score.
2. Determinar cuáles son las mejores métricas para el dataset obtenido y aplicarlas.
3. Desarrollar un módulo para calcular estas métricas en el modelo entrenado.
4. Realizar pruebas de validación utilizando los conjuntos de validación y prueba.
5. Generar informes de evaluación basados en las métricas calculadas.

**1.6.7. Objetivo 5: Desarrollar una aplicación web que permita desplegar los resultados del modelo**

1. Investigar sobre arquitecturas web y frameworks adecuados para el despliegue de aplicaciones de aprendizaje automático.
2. Diseñar una interfaz de usuario intuitiva que permita a los usuarios introducir datos y recibir predicciones del modelo.
3. Implementar una API que maneje las solicitudes de predicción y se comunique con el modelo de aprendizaje automático.

# Marco de referencia

---

## 2.1. Áreas Temáticas

Los códigos de clasificación utilizados se basan en el Sistema de Clasificación Computacional de la ACM 2012, un esquema polijerárquico desarrollado para aplicaciones web semánticas [11].

[500]· **Medicina y Salud** · *Diagnóstico médico* ~ *Trastornos de la voz* ~ *Otorrinolaringología* ~ *Aprendizaje automático en medicina* ·

[300]· **Procesamiento de Señales de Audio** · *Procesamiento de señales de voz* ~ *Análisis de señales de audio* ~ *Reconocimiento de patrones en señales de audio* ·

[300]· **Aprendizaje Automático y Deep Learning** · *Redes Neuronales Convolucionales (CNN)* ~ *Redes Neuronales Recurrentes (RNN)* ~ *Clasificación de datos secuenciales y temporales* ~ *Optimización de modelos de aprendizaje automático* ·

[300]· **Tecnología de la Información y Computación** · *Herramientas de procesamiento de datos* ~ *Desarrollo de modelos de aprendizaje profundo* ~ *Aplicaciones de aprendizaje automático en salud* ·

[300]· **Educación y Formación Profesional** · *Impacto de las patologías de la voz en la educación* ~ *Entrenamiento de profesionales de la voz* ·

[300]· **Ingeniería de Software y Desarrollo de Aplicaciones** · *Desarrollo de una API para mostrar resultados* ~ *Desarrollo de herramientas de procesamiento de señales de voz* ·

[100]· **Procesamiento de Datos Clínicos** · *Clasificación de datos clínicos* ~ *Validación de resultados en medicina* ·

## 2.2. Marco Teórico

### 2.2.1. Tipos de Aprendizaje

Se pueden categorizar ampliamente en tres tipos de aprendizaje: Supervisado, No Supervisado y Semi-supervisado. Cada tipo tiene sus méritos y casos de uso, y se ha demostrado que pueden mejorarse los resultados mediante el uso de aprendizaje no supervisado con secuencias de video no estructuradas [12].

### 2.2.2. Aprendizaje Supervisado

Dentro de las categorías del aprendizaje automático, existe el aprendizaje supervisado, el cual, se caracteriza por necesitar un conjunto de características o atributos de entrada y un conjunto de salida (conocido comúnmente como etiquetas) para cada entrada. Es decir, existen pares de entradas y salidas correctas, la entrada representa el objeto del estudio, caracterizado por ser un conjunto de variables y la salida es la etiqueta o clase que se quiere predecir [13]. Por ejemplo, en un contexto médico, las entradas serían los síntomas e indicadores de salud del paciente y la salida sería el diagnóstico.

#### 2.2.2.1. Regresión logística

La regresión logística es un modelo de aprendizaje automático basado en una función logística o sigmoidea que se utiliza para modelar la probabilidad de una variable binaria. La función logística está definida como:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

donde:

- -  $p$  es la probabilidad de la clase positiva.
- -  $\beta_0$  es una constante.
- -  $\beta_1, \beta_2, \dots, \beta_n$  son los coeficientes de regresión.
- -  $X_1, X_2, \dots, X_n$  son las características que ayudan en la predicción.

#### Hiperparámetros

Los hiperparámetros presentes en la regresión logística son pocos debido a la sencillez en sí misma del método. La tabla 2.1 presenta una descripción de los hiperparámetros de la regresión logística [14].



Hiperparámetro	Descripción
Regularización ( <i>penalty</i> )	Determina el tipo de regularización a aplicar: L1 (Lasso), L2 (Ridge) o Elastic Net. Ayuda a prevenir el sobreajuste.
Fuerza de Regularización ( <i>C</i> )	Inverso de la fuerza de regularización. Valores más pequeños especifican una regularización más fuerte.
Algoritmo de Optimización ( <i>solver</i> )	Algoritmo utilizado para encontrar los coeficientes de la regresión. Ejemplos incluyen: 'liblinear', 'saga', 'lbfgs', entre otros.
Máximo de Iteraciones ( <i>max_iter</i> )	Número máximo de iteraciones que el algoritmo de optimización utilizará para converger.

Cuadro 2.1: Hiperparámetros de la Regresión Logística

### 2.2.2.2. K-Nearest Neighbors (KNN)

El algoritmo K-Nearest Neighbors (KNN) es una técnica de clasificación sencilla y efectiva basada en la distancia entre los puntos de datos. Este algoritmo es sensible ante la normalización de los datos, ya que como bien se mencionó, hace uso de la distancia como sesgo de clasificación. KNN funciona calculando las distancias entre un punto de prueba y todos los puntos de entrenamiento, y luego asignando la clase más común entre los K vecinos más cercanos.

Usualmente el algoritmo de KNN hace uso de la distancia euclidiana para calcular la distancia entre el punto de entrada ( $y$ ) y todos los puntos existentes.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

### Hiperparámetros

La tabla 2.2 describe algunos de los hiperparámetros del algoritmo KNN.

Hiperparámetro	Descripción
<i>n_neighbors</i>	Número de vecinos a considerar (valor de $K$ ).
<i>weights</i>	Función de peso utilizada en la predicción. Puede ser 'uniform' o 'distance'.
<i>metric</i>	Métrica de distancia utilizada. Ejemplos: 'euclidean', 'manhattan', 'minkowski'.
<i>algorithm</i>	Algoritmo utilizado para computar los vecinos más cercanos. Opciones: 'auto', 'ball_tree', 'kd_tree', 'brute'.
<i>p</i>	Potencia del parámetro de la métrica de Minkowski.

Cuadro 2.2: Hiperparámetros comunes en K-Nearest Neighbors

### 2.2.2.3. Árboles de decisión

Los árboles de decisión son una forma de clasificador que se utiliza para etiquetar elementos de datos con uno de un conjunto finito de clases, basándose en características asociadas a estos datos. Estos clasificadores funcionan mediante la formulación de una serie de preguntas, cada una representada por un nodo en el árbol, donde las respuestas guían el proceso de clasificación desde el nodo raíz hasta una hoja que proporciona la etiqueta de clasificación [15]. Esta estructura jerárquica permite que los árboles de decisión manejen de manera intuitiva la toma de decisiones secuenciales y condicionales.

La construcción de un árbol de decisión se basa en un conjunto de ejemplos de entrenamiento con etiquetas conocidas. A través de un proceso iterativo, el árbol crece añadiendo nodos de pregunta que estratifican los datos de entrenamiento. El objetivo es dividir los datos de manera que los nodos finales del árbol, o las hojas, contengan ejemplos lo más homogéneos posible en cuanto a la etiqueta de clase. Para evaluar la calidad de las divisiones y elegir las preguntas, se utilizan medidas como la entropía o el índice de Gini, que cuantifican la impureza de los nodos y buscan minimizarla [15].

### Hiperparámetros

La tabla 2.3 presenta algunos de los hiperparámetros de los árboles de decisión.

Hiperparámetro	Descripción
<i>criterion</i>	Función para medir la calidad de una división. Ejemplos: 'gini', 'entropy'.
<i>splitter</i>	Estrategia utilizada para elegir la división en cada nodo. Opciones: 'best', 'random'.
<i>max_depth</i>	Profundidad máxima del árbol.
<i>min_samples_split</i>	Número mínimo de muestras necesarias para dividir un nodo.
<i>min_samples_leaf</i>	Número mínimo de muestras necesarias para ser una hoja.
<i>max_features</i>	Número máximo de características a considerar para encontrar la mejor división.

Cuadro 2.3: Hiperparámetros comunes en Árboles de Decisión

#### 2.2.2.4. Random Forest

El algoritmo de Random Forest se ha consolidado como una de las técnicas más robustas y eficaces para los problemas de aprendizaje supervisado de clasificación y regresión. La esencia del algoritmo es construir un conjunto de árboles de decisión, cada uno de ellos se entrena con un vector aleatorio independiente, que sigue la misma distribución para todo el bosque. El algoritmo se beneficia de la capacidad de aprendizaje individual de cada árbol, realzando sus características, así como también se beneficia de la diversidad de árboles, permitiendo controlar el sobreajuste y la precisión de predicción del modelo [16].

#### Hiperparámetros

La tabla 2.4 presenta algunos de los hiperparámetros del algoritmo de random forest.

Hiperparámetro	Descripción
<i>n_estimators</i>	Número de árboles en el bosque.
<i>max_features</i>	Número máximo de características a considerar para encontrar la mejor división.
<i>max_depth</i>	Profundidad máxima de los árboles.
<i>min_samples_split</i>	Número mínimo de muestras necesarias para dividir un nodo.
<i>min_samples_leaf</i>	Número mínimo de muestras necesarias para ser una hoja.
<i>bootstrap</i>	Método para muestrear datos al entrenar cada árbol. Puede ser True o False.
<i>criterion</i>	Función para medir la calidad de una división. Ejemplos: 'gini', 'entropy'.

Cuadro 2.4: Hiperparámetros comunes en Random Forest

En resumen, Random Forest emerge como una metodología poderosa dentro del aprendizaje automático, destacando por su capacidad de sintetizar múltiples árboles de decisión para mejorar la precisión y la estabilidad de las predicciones. Su diseño inherente combate el sobreajuste y proporciona herramientas para la interpretación del modelo, lo que lo hace invaluable para aplicaciones prácticas en campos tan variados como la medicina, la banca y más allá.

### 2.2.3. Aprendizaje profundo

El aprendizaje profundo es una subclase de métodos de aprendizaje automático que ha ganado tracción en una variedad de aplicaciones. A diferencia de los enfoques tradicionales que requieren un extractor de características diseñado manualmente, el aprendizaje profundo es capaz de aprender automáticamente buenas características directamente desde datos brutos [17]. Este enfoque se basa en el uso de redes neuronales artificiales con múltiples capas (también conocidas como redes neuronales profundas), que permiten modelar y entender datos complejos de manera más efectiva. Entre las aplicaciones más comunes del aprendizaje profundo se encuentran el reconocimiento de voz, la visión por computadora, la traducción automática y la conducción autónoma, entre otras.

#### 2.2.3.1. Arquitectura y Aprendizaje

Una arquitectura de aprendizaje profundo consiste en una pila de módulos simples, cada uno de los cuales es objeto de aprendizaje. Estos módulos computan mapeos de entrada-salida no lineales y están diseñados para aumentar tanto la selectividad como la invarianza de la representación. Con múltiples capas no lineales, un sistema puede implementar funciones extremadamente intrincadas de sus entradas que son simultáneamente sensibles a detalles minúsculos e insensibles a variaciones irrelevantes, como el fondo, la pose y la iluminación [17].

### 2.2.3.2. Componentes de la Arquitectura

1. **Capas Convolucionales (Conv Layers):** Utilizadas principalmente en el procesamiento de imágenes, estas capas aplican filtros para extraer características locales de la entrada, como bordes, texturas y patrones.
2. **Capas de Pooling:** Reducen la dimensionalidad de las características extraídas por las capas convolucionales, ayudando a disminuir el costo computacional y a controlar el sobreajuste.
3. **Capas Densas (Fully Connected Layers):** Cada neurona en una capa está conectada a todas las neuronas de la capa anterior, permitiendo la integración de características aprendidas para la toma de decisiones finales.
4. **Capas Recurrentes (Recurrent Layers):** Comúnmente usadas en datos secuenciales, como series temporales o texto, estas capas tienen conexiones de retroalimentación que permiten mantener información temporal.

### 2.2.3.3. Elementos del Aprendizaje

El aprendizaje profundo emplea diversas técnicas para optimizar y ajustar los modelos de redes neuronales. A continuación, se detallan algunas de las técnicas comunes utilizadas en el entrenamiento de redes neuronales:

1. **Backpropagation:** El algoritmo de backpropagation, o retropropagación del error, es una técnica fundamental para entrenar redes neuronales. Permite ajustar los pesos de la red neuronal mediante la minimización del error en la salida [18]. Este proceso se lleva a cabo en múltiples iteraciones conocidas como epochs. Durante cada epoch, se calcula el error de la red, y luego se propaga hacia atrás a través de la red para actualizar los pesos en la dirección opuesta al gradiente del error. La fórmula general para actualizar un peso  $w$  es:

$$w_{new} = w_{old} - \eta \frac{\partial E}{\partial w}$$

donde  $\eta$  es la tasa de aprendizaje y  $\frac{\partial E}{\partial w}$  es el gradiente del error  $E$ . Este proceso asegura que la red se ajuste gradualmente para reducir el error y mejorar su precisión.

2. **Regularización:** La regularización es crucial para evitar el sobreajuste (overfitting) del modelo, que ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos. Se emplean varias técnicas de regularización, entre las cuales destacan:
  - **Dropout:** Consiste en apagar aleatoriamente un porcentaje de neuronas durante el entrenamiento. Esto previene que el modelo dependa excesivamente de ciertas neuronas y fomenta la redundancia y la robustez en la red. Durante cada iteración de entrenamiento, las neuronas seleccionadas al azar se eliminan temporalmente del proceso de aprendizaje [19].

- **L2 Regularization** (también conocido como weight decay): Penaliza grandes pesos añadiendo un término al error de la función objetivo. La función de costo regularizada se define como:

$$E_{reg} = E + \lambda \sum_i w_i^2$$

donde  $\lambda$  es el parámetro de regularización y  $w_i$  son los pesos del modelo. Esta técnica ayuda a mantener los pesos pequeños, lo que a su vez simplifica el modelo y reduce el riesgo de sobreajuste [20].

3. **Optimización:** Los algoritmos de optimización son esenciales para ajustar los pesos de la red de manera eficiente durante el entrenamiento. Un algoritmo popular es **Adam** (Adaptive Moment Estimation), que combina las ventajas de los algoritmos AdaGrad y RMSProp. Adam ajusta los pesos utilizando los primeros y segundos momentos del gradiente, lo que permite actualizaciones adaptativas de los pesos. Las fórmulas utilizadas en Adam son:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

donde  $g_t$  es el gradiente en el tiempo  $t$ ,  $m_t$  y  $v_t$  son los promedios móviles del primer y segundo momento del gradiente, respectivamente, y  $\beta_1$  y  $\beta_2$  son los parámetros de control de decaimiento exponencial. Adam es conocido por su eficiencia y capacidad de manejar grandes conjuntos de datos y altos números de parámetros [21].

#### 2.2.3.4. Clasificación General de Redes Neuronales

Las **Convolutional Neural Networks (CNNs)** son una clase de redes neuronales especialmente diseñadas para procesar datos con una estructura de rejilla, como las imágenes y secuencias temporales. Han demostrado un rendimiento superior en tareas de visión por computadora y han sido adaptadas para una variedad de aplicaciones en diferentes dominios [22]. A continuación, se detalla su estructura y funcionamiento.

1. **Capas Convolucionales:** Las capas convolucionales son el núcleo de las CNNs. Estas capas aplican filtros (kernels) sobre la entrada para producir mapas de características. Un filtro es una matriz de pesos que se desplaza (convoluciona) sobre la entrada, multiplicándose por

segmentos de la entrada y sumando los resultados para producir un valor en el mapa de características [22]. La operación de convolución se define como:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau$$

En el contexto de CNNs, se simplifica a:

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

donde  $I$  es la entrada y  $K$  es el kernel.

2. **Capas de Pooling:** Las capas de pooling reducen las dimensiones espaciales de los mapas de características, permitiendo la disminución del número de parámetros y computación en la red, además de proporcionar invariancia a traslaciones [22]. Los tipos más comunes son el max pooling y el average pooling. Max pooling selecciona el valor máximo de una región (normalmente 2x2), mientras que average pooling calcula el promedio. La fórmula para max pooling es:

$$P(i, j) = \max_{(m,n) \in R} X(i + m, j + n)$$

donde  $R$  es la región del pooling.

3. **Capas de Normalización:** Estas capas normalizan los valores en los mapas de características para acelerar el entrenamiento y mejorar la estabilidad del modelo [22]. Un tipo común es Batch Normalization, que normaliza las activaciones de una capa utilizando la media y desviación estándar del minibatch:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

donde  $\mu$  y  $\sigma^2$  son la media y la varianza del minibatch, respectivamente, y  $\varepsilon$  es un término pequeño para evitar divisiones por cero.

4. **Capas de Activación:** Las capas de activación introducen no linealidades en la red, permitiendo que la red aprenda relaciones complejas [22]. Las funciones de activación comunes incluyen ReLU (Rectified Linear Unit), que se define como:

$$f(x) = \max(0, x)$$

y otras variantes como Leaky ReLU, Tanh, y Sigmoid.

5. **Capas Densas (Fully Connected Layers):** Estas capas conectan cada neurona en una capa a todas las neuronas de la siguiente capa. Son utilizadas principalmente en las últimas etapas de la red, después de varias capas convolucionales y de pooling, para combinar las características extraídas y realizar la clasificación [22]. La operación se define como:

$$z = Wx + b$$

donde  $W$  es la matriz de pesos,  $x$  es el vector de entrada, y  $b$  es el vector de sesgos.

### 2.2.3.5. Ventajas y Aplicaciones de las CNNs

Las CNNs presentan varias ventajas que las hacen ideales para tareas de procesamiento de datos estructurados:

- **Invariancia Espacial:** Gracias a las capas de pooling, las CNNs pueden reconocer patrones independientemente de su posición en la entrada.
- **Extracción Automática de Características:** Las CNNs aprenden a extraer características relevantes de los datos de entrada, eliminando la necesidad de un preprocesamiento manual extenso.
- **Escalabilidad:** Las CNNs pueden manejar grandes cantidades de datos y son escalables a redes más profundas, mejorando su capacidad de aprendizaje con arquitecturas más complejas.

### 2.2.3.6. Arquitecturas de CNNs

Existen varias arquitecturas de CNNs que se utilizan en el procesamiento de datos:

1. **Simple CNN:** Una arquitectura básica de CNN compuesta por una secuencia de capas convolucionales, capas de pooling y capas densas finales para la clasificación.
2. **Deep CNN:** Arquitecturas más profundas que incluyen múltiples capas convolucionales y de pooling para capturar características más complejas y abstracciones de alto nivel en los datos.
3. **Transfer Learning con CNNs Preentrenadas:** Utilización de modelos preentrenados como VGG16, ResNet50, entre otros, que han sido entrenados previamente en grandes conjuntos de datos como ImageNet, y ajustados (fine-tuned) para tareas específicas.

Las técnicas y arquitecturas de CNNs han demostrado ser efectivas en diversas aplicaciones, mejorando el rendimiento en tareas complejas y permitiendo avances significativos en el campo del aprendizaje profundo [22].

### 2.2.4. Comparativa entre modelos

En la tabla 2.5 se presenta una comparativa en términos de ventajas y desventajas de los modelos y arquitecturas mencionados anteriormente.



Modelo	Ventajas	Desventajas
<b>Regresión Logística</b>	<ul style="list-style-type: none"> <li>- Fácil de implementar y entender</li> <li>- Buena interpretación de coeficientes</li> <li>- Eficiente para conjuntos de datos grandes</li> </ul>	<ul style="list-style-type: none"> <li>- Asume linealidad entre variables independientes y dependientes</li> <li>- No es adecuada para problemas no lineales</li> <li>- Sensible a outliers</li> </ul>
<b>K-Nearest Neighbors (KNN)</b>	<ul style="list-style-type: none"> <li>- Simple y fácil de entender</li> <li>- No requiere suposición sobre la distribución de datos</li> <li>- Funciona bien con datos no lineales</li> <li>- No requiere cómputo en entrenamiento</li> </ul>	<ul style="list-style-type: none"> <li>- Computacionalmente costoso para grandes conjuntos de datos, ya que se considera un algoritmo perezoso (lazy)</li> <li>- Sensible al ruido y a los outliers</li> <li>- Requiere elección de <math>k</math> adecuado</li> </ul>
<b>Árboles de Decisión</b>	<ul style="list-style-type: none"> <li>- Fácil de interpretar y visualizar</li> <li>- Maneja datos categóricos y continuos</li> <li>- No requiere mucha preparación de datos</li> </ul>	<ul style="list-style-type: none"> <li>- Propenso al sobreajuste</li> <li>- Inestable ante pequeñas variaciones en los datos</li> <li>- Puede ser sesgado si una clase domina</li> </ul>
<b>Random Forest</b>	<ul style="list-style-type: none"> <li>- Reduce el sobreajuste comparado con árboles de decisión individuales</li> <li>- Maneja grandes conjuntos de datos con alta dimensionalidad</li> <li>- Alta precisión</li> </ul>	<ul style="list-style-type: none"> <li>- Difícil de interpretar comparado con árboles de decisión simples</li> <li>- Más lento y computacionalmente costoso</li> <li>- Requiere ajuste de hiperparámetros</li> </ul>
<b>Redes Neuronales Convolucionales (CNN)</b>	<ul style="list-style-type: none"> <li>- Excelente para procesamiento de imágenes y datos espaciales</li> <li>- Capacidad para aprender características complejas</li> <li>- Alta precisión en problemas complejos</li> </ul>	<ul style="list-style-type: none"> <li>- Requiere grandes volúmenes de datos para entrenar</li> <li>- Computacionalmente intensivo</li> <li>- Difícil de interpretar y ajustar</li> </ul>

Cuadro 2.5: Ventajas y Desventajas de los Modelos Utilizados

### 2.2.5. Evaluación y Validación

Para asegurar la efectividad de los modelos de aprendizaje automático, se utilizan diversas técnicas de validación y métricas de evaluación. Estas técnicas permiten no solo entrenar el modelo, sino también evaluar su rendimiento de manera objetiva y robusta.

1. **Cross-Validation:** La validación cruzada (cross-validation) es una técnica utilizada para evaluar el rendimiento del modelo de manera más robusta. Esta técnica divide los datos en múltiples subconjuntos (folds) y entrena el modelo  $k$  veces, cada vez utilizando un subconjunto diferente como datos de prueba y los otros  $k - 1$  subconjuntos como datos de entrenamiento. En  $k$ -fold cross-validation, por ejemplo, los datos se dividen en  $k$  partes, y el proceso se repite  $k$  veces, asegurando que cada parte se use como conjunto de prueba exactamente una vez. El rendimiento final se obtiene promediando los resultados de cada iteración, lo que proporciona una estimación más confiable de la capacidad del modelo para generalizar a nuevos datos [23].
2. **Métricas de Rendimiento:** Para evaluar y comparar el desempeño de los modelos, se emplean varias métricas de rendimiento, entre las cuales se incluyen:

- **Accuracy:** La precisión (accuracy) es la proporción de predicciones correctas entre el total de predicciones. Se define como:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

donde  $TP$  es el número de verdaderos positivos,  $TN$  es el número de verdaderos negativos,  $FP$  es el número de falsos positivos y  $FN$  es el número de falsos negativos. Esta métrica es útil para evaluar modelos en datasets balanceados.

- **Precision:** La precisión (precision) es la proporción de verdaderos positivos entre el total de predicciones positivas. Se define como:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Esta métrica es crucial en situaciones donde el costo de un falso positivo es alto.

- **Recall:** El recall (también conocido como sensibilidad) es la proporción de verdaderos positivos entre el total de casos positivos reales. Se define como:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Esta métrica es importante cuando es crítico capturar todos los casos positivos.

- **F1-Score:** El F1-score es la media armónica de precision y recall, proporcionando una medida equilibrada del rendimiento del modelo. Se define como:

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

El F1-score es útil en datasets desbalanceados, donde ni la precision ni el recall por sí solos proporcionan una evaluación completa del rendimiento del modelo [24].

- **Matriz de Confusión:** Una matriz de confusión es una herramienta que permite visualizar el rendimiento del modelo de clasificación, mostrando la cantidad de predicciones correctas e incorrectas desglosadas por cada clase [24]. Un ejemplo de matriz de confusión es:

	Predicho Positivo	Predicho Negativo
Real Positivo	TP	FN
Real Negativo	FP	TN

Cuadro 2.6: Matriz de Confusión

La matriz de confusión proporciona una representación clara de cómo el modelo se desempeña en términos de cada clase, permitiendo identificar áreas específicas donde el modelo puede estar fallando.

Las técnicas mencionadas ayudan a mejorar el rendimiento y la generalización de los modelos de aprendizaje profundo, asegurando que sean precisos, robustos y adecuados para una variedad de aplicaciones prácticas.

## 2.2.6. Lectura de muestras de audio.

### 2.2.6.1. Amplitud de onda en instantes discretos

La técnica de procesamiento de amplitudes de onda se centra en analizar las amplitudes de una señal de audio en función del tiempo. Este método se basa en muestrear la señal de audio a intervalos regulares, lo que permite obtener una representación detallada de la variación de la amplitud a lo largo del tiempo. La señal de audio, una vez muestreada, se convierte en una serie de valores numéricos que representan la intensidad de la señal en cada instante de muestreo ??.

Este enfoque permite realizar un análisis temporal preciso de la señal, identificando características específicas y eventos transitorios que ocurren en diferentes momentos. La amplitud de la señal en cada punto proporciona información valiosa sobre la energía y dinámica de la señal de audio.

El procesamiento de amplitudes de onda es usado para diversas aplicaciones en el ámbito del procesamiento de audio, como la detección de eventos acústicos, la clasificación de señales y el reconocimiento de patrones. Esta técnica es directa y proporciona una representación fiel de la señal original, lo que facilita su análisis y manipulación sin necesidad de transformaciones complejas.

### 2.2.6.2. Transformada de Fourier de Tiempo Corto y sus Beneficios para el Trabajo con Audios

La Transformada de Fourier de Tiempo Corto (STFT, por sus siglas en inglés) es una técnica fundamental en el análisis de señales de audio, especialmente útil para señales no estacionarias como las de audio. Esta técnica descompone una señal en sus componentes de frecuencia a lo largo del tiempo mediante la aplicación de la Transformada de Fourier a segmentos cortos de la señal. Matemáticamente, la STFT de una señal  $x(t)$  se define como:

$$\text{STFT}(x(t)) = X(t, f) = \int_{-\infty}^{\infty} x(\tau) w(t - \tau) e^{-j2\pi f\tau} d\tau \quad (2.1)$$

donde  $w(t)$  es una ventana deslizando que selecciona un segmento de la señal en cada instante de tiempo  $t$ , y  $f$  representa la frecuencia. La elección de la ventana  $w(t)$  y su ancho son críticos, ya que determinan la resolución tiempo-frecuencia del análisis.

La STFT se utiliza para generar espectrogramas, representaciones visuales de la densidad espectral de potencia de una señal en función del tiempo y la frecuencia, esenciales para visualizar y analizar la estructura temporal y frecuencial de señales de audio complejas. En el ámbito de la clasificación de audio, la STFT permite extraer características robustas de las señales, mejorando el rendimiento de los modelos de clasificación. Por ejemplo, en estudios comparativos, se ha demostrado que las características extraídas mediante STFT pueden ser más efectivas que otras técnicas, como los coeficientes cepstrales en frecuencia Mel (MFCC), especialmente en entornos con ruido [25].

La capacidad de la STFT para proporcionar una representación detallada tiempo-frecuencia facilita la detección de eventos acústicos específicos, como explosiones, alarmas y otros sonidos de interés en aplicaciones de vigilancia y seguridad [25]. Además, la STFT es útil en técnicas de reducción de ruido, permitiendo identificar y filtrar componentes de ruido en diferentes bandas de frecuencia sin afectar significativamente las componentes de señal deseadas, lo cual es beneficioso en la limpieza y mejora de la calidad de grabaciones de audio en entornos ruidosos. También se emplea en sistemas de reconocimiento de patrones y reconocimiento de voz, proporcionando una base sólida para algoritmos que necesitan identificar características específicas de las señales acústicas, siendo especialmente adecuada para el análisis de voz y otros sonidos dinámicos [25].

### 2.2.7. Patologías en la voz.

Una patología de la voz se define como cualquier alteración o desorden que afecte la producción normal de la voz. Esto incluye cualquier cambio en la calidad, tono, intensidad o duración de la voz que resulte de condiciones anormales en las estructuras involucradas en la producción vocal, como las cuerdas vocales o los órganos de articulación [7]. Las patologías vocales pueden ser causadas por una variedad de factores, incluyendo uso excesivo de la voz, infecciones, trastornos neurológicos, problemas psicológicos o condiciones médicas subyacentes. Estas alteraciones pueden impactar significativamente la comunicación de los individuos afectados.

#### 2.2.7.1. Ejemplos de patologías en la voz.

1. **Laringitis Aguda:** Se caracteriza por la inflamación de la capa superficial de la lámina propia de las cuerdas vocales, lo que puede causar edema o rigidez. Esto altera el patrón vibratorio de las cuerdas vocales, afectando la calidad de la voz [4].
2. **Nódulos Vocales:** Son pequeñas lesiones blanquecinas, habitualmente bilaterales, ubicadas en las cuerdas vocales. Estos nódulos pueden interferir con el cierre completo de la glotis y varían en su presentación, dependiendo de si son edematosos o fibrosos [4].

3. **Pólipos Laríngeos:** Pueden variar en color y tamaño. Se desarrollan en el borde libre de las cuerdas vocales. Afectan la capacidad de cierre glótico y pueden causar una vibración asimétrica y aperiódica de las cuerdas [4].
4. **Quistes Vocales:** Son lesiones que pueden ser congénitas o adquiridas, ubicadas en la capa superficial de la lámina propia. Alteran las propiedades mecánicas de las cuerdas vocales y pueden afectar el cierre glótico [4].

## 2.3. Trabajos Relacionados

### 2.3.1. Automatic Assessment of Voice Quality in the Context of Multiple Annotations [1]

Este proyecto se enfoca en la evaluación automática de la calidad del habla mediante análisis perceptivo y acústico, considerando múltiples anotaciones de expertos. Se comparan dos técnicas de clasificación, una basada en procesos gaussianos y otra en regresión logística multiclase, utilizando el índice Kappa de Cohen para evaluar el rendimiento. Los resultados muestran que los esquemas con múltiples anotaciones superan a las técnicas tradicionales.

Esto guarda relación con la iniciativa propuesta, ya que también se busca evaluar la calidad del habla en la detección de patologías y se pueden obtener ideas útiles de este trabajo.

### 2.3.2. A review on voice pathology: Taxonomy, diagnosis, medical procedures and detection techniques, open challenges, limitations, and recommendations for future directions [2]

Este artículo se centra en la revisión de las patologías de la voz, incluyendo su taxonomía, diagnóstico, procedimientos médicos, técnicas de detección, desafíos abiertos, limitaciones y recomendaciones para futuras direcciones. Se destaca la importancia de la detección temprana de problemas de voz y cómo los procedimientos basados en computadoras, en lugar de los métodos tradicionales, pueden ser más eficientes y accesibles.

La relación de este artículo con el proyecto propuesto radica en que proporciona enfoques y metodologías útiles para mejorar los métodos de evaluación de la calidad de la voz y la detección de patologías vocales.

### 2.3.3. Identification of Pathological Voices Using Glottal Noise Measures [3]

El artículo de investigación de Vijay Parsa y Donald G. Jamieson se centra en la evaluación de diferentes medidas para cuantificar el ruido vocal. En el estudio, se presentan diversas métricas, tanto dependientes como independientes de la frecuencia fundamental ( $F_0$ ), y se discuten sus aplicaciones y limitaciones. Se explora en detalle las ventajas y desventajas de cada medida, proporcionando una visión integral que podría ser de gran utilidad para la identificación y tratamiento

de desórdenes vocales.

En el contexto del trabajo propuesto, este artículo es particularmente relevante debido a la base de datos clasificada que postula y a los métodos que utiliza para realizar la exploración y de los datos.

# Conjunto de Datos

---

## 3.1. Recolección de datos.

Para el presente trabajo se hizo uso de un subconjunto de la base de datos de Massachusetts Eye and Ear Infirmary Disordered Voice Database de por la industria Kay Elemetrics. Específicamente se tomó un subconjunto de 226 muestras de voz descritas por [3]. Las muestras de voz corresponden a la fonación sostenida de la vocal 'a'. Cada registro de voz fue dividido en saludable o no saludable según corresponda clasificado con la escala GRBAS por múltiples expertos en el área.

### 3.1.1. Especificación de los datos.

Las muestras de voz tienen una duración que varía entre 1 y 3 segundos. La duración promedio de las muestras es de 57170 frames, lo que en segundos equivaldría a una tasa de muestreo de 50000Hz a 1.46 segundos, la duración promedio de las muestras patológicas es de 1 segundo y la duración promedio de las muestras sanas es de 3 segundos.

Existe un total de 226 muestras de voz divididas en:

- 167 catalogadas como no saludables.
- 53 catalogadas como saludables.

El conjunto de datos ha sido utilizado en una amplia gama de trabajos asociados al reconocimiento de patologías en la voz a través de, en un inicio, artificios matemáticos y estadísticos y posteriormente de aprendizaje automático.

Sin embargo, como se observa en la gráfica 3.1 la proporción de los datos es de un 73.9% de muestras asociadas a una patología y un 26.1% de muestras saludables. Lo que deja una distribución sesgada hacia las muestras patológicas, existiendo así una diferencia del 47.8% entre ambas clases. Cabe recalcar que este sesgo puede influir en los resultados obtenidos.

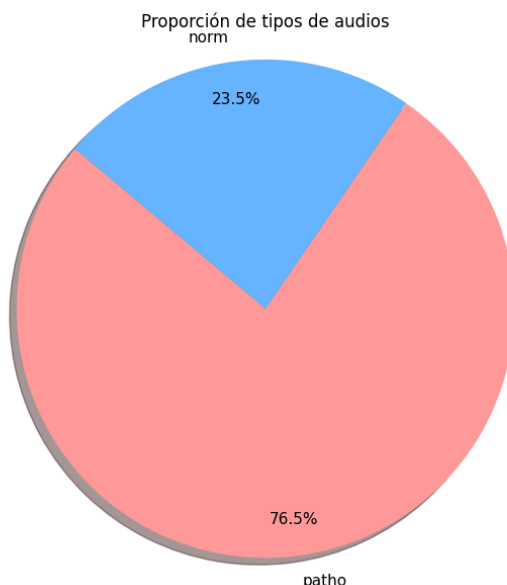


Figura 3.1: Proporción de tipos de audios

### 3.2. Lectura de las muestras.

La lectura de las muestras de audio para esta fase se realizó utilizando la librería de Python wave, que proporciona una interfaz para leer y escribir archivos WAV (formato de audio sin comprimir). Esta librería permite acceder a los datos de audio en su formato binario original.

Para la lectura de las muestras se utilizó la función `wave.open` que abre los audios almacenados en el disco en modo lectura binaria. Los datos de audio se leen en formato binario y se convierten en un array de tipo entero de 16 bits, que es el formato estándar de muestras de audio sin comprimir. Estas muestras de audio se normalizan a valores de punto flotante en el rango  $[-1.0, 1.0]$  dividiendo por 32767.0; que corresponde al valor máximo de un entero de 16 bits. Finalmente, el resultado de esta operación da como resultado un arreglo donde cada elemento corresponde a la amplitud de la señal de audio en un instante específico de tiempo. Por lo tanto, puede entenderse que si  $f$  es la función de onda del audio, todos los elementos del dominio corresponden a la duración del audio en instantes discretos, los cuales a su vez están determinados por la frecuencia de muestreo. Todos los elementos del rango son los valores de la amplitud de la onda en esos instantes de tiempo. En conclusión,  $f(t)$  da como resultado el valor de la amplitud de la onda en el instante de tiempo discreto  $t$ .

El código de Python que desarrolla lo descrito anteriormente es presentado a continuación:

```
1 def readAudioFiles( pathAudioType, df, tipo ):  
2     for file in os.listdir( path + pathAudioType ):  
3         with wave.open( path + pathAudioType + file, 'rb' ) as audioFile:  
4             amountFrames = audioFile.getnframes()  
5             datosAudio = audioFile.readframes( amountFrames )
```



```

6     audioArray = np.frombuffer( datosAudio, dtype=np.int16 ).astype( np.
    float32 ) / 32767.0
7     dfNew = pd.DataFrame( { 'audio': [ audioArray ], 'type': [ tipo ] } )
8     df = pd.concat( [ df, dfNew ], ignore_index = True )
9     return df

```

Listing 3.1: Lectura de datos usando wave

### 3.3. Preprocesamiento técnicas de aprendizaje clásicas.

El conjunto de datos requiere un preprocesamiento dada la discrepancia en la duración promedio de las muestras de voz. Cabe recalcar que la transformación debe ser aplicada de manera uniforme a todas las muestras. La cantidad de información que podría perderse dada la normalización, es en parte, contrarrestada por la forma en la que se grabaron las mismas, donde se utiliza enteramente y únicamente el fonema 'a' a lo largo de la muestra.

La descripción específica de las tareas realizadas durante la normalización del conjunto de datos se presentan a continuación:

1. Determinar cual es la duración promedio de las muestras de voz en el dataset.
2. Normalizar la duración dejando todas las muestras en una duración promedio. Este punto se divide a su vez en dos más:
  - Si la duración de la muestra de voz es mayor al promedio, entonces, se debe recortar una parte de la muestra para que tenga la misma duración.
  - Si la duración de la muestra de voz es menor al promedio, entonces, se debe rellenar con ceros para completar la duración promedio.

Sin embargo, existen otras alternativas en cuanto al proceso de normalización en este punto. Por ejemplo, se puede obtener el audio con la duración mínima y recortar el resto de audios a esta duración, sin embargo, ya que la duración mínima es de 11263 frames y con base en la duración promedio de las muestras que es de 57170 frames, se estarían perdiendo, en promedio 45907 frames que pueden contribuir de manera significativa al entrenamiento de los modelos de aprendizaje y pueden contener características esenciales para la detección de las patologías.

En la tabla 3.1 se presentan las características finales del dataset posterior a su normalización.

#	Column	Non-Null Count	Dtype
0	audio	226 non-null	object
1	type	226 non-null	object

Cuadro 3.1: Descripción de las columnas del dataset

### 3.4. Preprocesamiento técnicas de aprendizaje aprendizaje profundo

Para los modelos de aprendizaje profundo se optó por usar otro enfoque como el descrito en [25] en el que se transforman las señales de voz en imágenes que corresponden a espectrogramas de las muestras. Con estas imágenes son entrenados los modelos de aprendizaje profundo en lugar de tener un arreglo que corresponde a la amplitud de la onda. Los pasos para la transformación de las muestras de audio en espectrogramas son descritos a continuación:

1. **Carga del Archivo de Audio:** La señal de audio se carga desde el archivo, junto con su frecuencia de muestreo.
2. **Transformación de Fourier de Corto Tiempo (STFT):** Se aplica la Transformación de Fourier de Corto Tiempo (STFT) a la señal de audio para convertirla en un espectrograma, que muestra cómo varía la frecuencia con el tiempo.
3. **Conversión a Escala de Decibelios:** El espectrograma se convierte en una escala de decibelios.
4. **Visualización y Guardado del Espectrograma:** Se visualiza y guarda la imagen del espectrograma, configurando los ejes del tiempo y la frecuencia en una escala logarítmica.
5. **Procesamiento de Archivos de Audio:** Se itera a través de los archivos de audio en el directorio. Para cada archivo de audio, se guarda la imagen del espectrograma en el subdirectorio de imágenes.

Cabe aclarar que la Transformada de Fourier de Corto Tiempo (STFT), como lo menciona Gourisaria *et al.* [25], es un artificio matemático fuertemente utilizado para extraer características de audios. Lo que realiza la Transformada de Fourier de Corto Tiempo es dividir la señal de audio en segmentos cortos y aplicar la Transformada de Fourier a cada segmento para analizar cómo varían las frecuencias en el tiempo. Es importante recordar que la Transformada de Fourier es una técnica matemática que convierte una señal en función del tiempo en una representación en función de la frecuencia.

Una ventaja importante de utilizar la Transformada de Fourier de Corto Tiempo (STFT) frente a otras técnicas es su simplicidad matemática y eficiencia en el procesamiento, lo que la convierte en una opción especialmente adecuada para tareas de análisis de audio en tiempo real o de procesamiento continuo. A diferencia de métodos como la transformada wavelet o la transformada de Hilbert-Huang, que pueden requerir una mayor complejidad computacional y ajustes específicos de parámetros, la STFT permite obtener una representación consistente y clara de las frecuencias en segmentos cortos de la señal sin complicaciones adicionales. Además, su uso está ampliamente soportado en diversas herramientas y bibliotecas, facilitando su implementación en múltiples entornos y aplicaciones de procesamiento de audio, lo cual contribuye a su popularidad en el ámbito del análisis de señales. Adicional a ello, la literatura existente referente a la transformada de Fourier es

muy grande y ha demostrado tener un amplio uso en cuanto a labores de procesamiento de señales de audio se refiere.

Para la lectura de las imágenes correspondientes a la representación de los audios como espectrogramas se utilizó la librería de Python, la cual es comunmente utilizada para tareas que involucran imágenes, de tal forma que las imágenes se convirtieron en matrices de cuádruplas donde cada elemento de la cuádrupla está en una de 0 a 255 correspondiente al esquema de colores RGB y el cuarto elemento corresponde al canal alpha o brillo. Las imágenes son almacenadas en un dataset diferente al de la lectura de las señales de audio definido anteriormente, cada imagen tiene las dimensiones de 308\*775 píxeles. El dataset de imágenes es presentado en la tabla 3.2 junto con las dimensiones de las imágenes.

Característica	Valor
Número total de imágenes	226
Dimensiones de las imágenes	(308, 775, 4)
Número de imágenes normales	167
Número de imágenes patológicas	53

Cuadro 3.2: Características del dataset de imágenes

#### 3.4.1. Aumento de datos técnicas clásicas.

Dada la cantidad limitada de datos en el presente proyecto, se optó por realizar una técnica conocida como aumento de datos (data aumengtation), la cual consiste principalmente en generar datos a partir de los datos existentes transformados de diversas formas [26]. Por ejemplo, una posible transformación sería invertir una señal de audio. Esto se puede realizar aplicando una inversión temporal, donde la señal se reproduzca al revés e implementarlo sería tan sencillo como invertir el arreglo que contiene los valores de la amplitud. Otra posible técnica es cambiar la velocidad de reproducción del audio sin alterar su tono.

En el presente trabajo para el aumento de datos se realizaron los pasos mencionados a continuación:

- **Definición del desplazamiento temporal:**

- Se aplica un desplazamiento temporal a las señales de audio, lo cual consiste en mover la señal hacia adelante o hacia atrás en el tiempo.
- Si el desplazamiento es positivo, se añaden ceros al inicio de la señal y se eliminan los últimos valores de la misma.
- Si el desplazamiento es negativo, se eliminan los primeros valores de la señal y se añaden ceros al final de la misma.

- **Determinación del porcentaje de desplazamiento:**

- Se establece un porcentaje máximo de desplazamiento con respecto al tamaño total de la señal de audio.
- Este porcentaje define el rango dentro del cual se seleccionará aleatoriamente el valor del desplazamiento para cada señal.

■ **Generación del desplazamiento:**

- Para cada señal de audio en el conjunto de datos original, se genera un valor de desplazamiento aleatorio dentro del rango determinado.
- Este valor de desplazamiento se aplica a la señal, creando una versión desplazada de la misma.

Por lo tanto, al final de este proceso el resultado es un dataset con el doble de datos que el que se tenía originalmente. Es decir, inicialmente se tenía 226 muestras de audio y después de realizar el aumento de datos se obtuvieron 226 muestras más generando un total de 452 muestras de voz. La tabla 3.3 muestra las características del dataset con aumento de datos.

#	Column	Non-Null Count	Dtype
0	audio	452 non-null	object
1	type	452 non-null	object

Cuadro 3.3: Descripción de las columnas del dataset aumentado

El fragmento de código que ejecuta y describe la tarea descrita anteriormente en Python es presentado a continuación:

```

1 def timeShift(audioArray, shift):
2     if shift > 0:
3         return np.concatenate([np.zeros(shift), audioArray[:-shift]])
4     elif shift < 0:
5         return np.concatenate([audioArray[-shift:], np.zeros(-shift)])
6     return audioArray
7
8 def applyTimeShiftAugmentation(df, maxShiftPct=0.1):
9     dfAugmented = pd.DataFrame(columns=['audio', 'type'])
10
11     for index, row in df.iterrows():
12         audioArray = row['audio']
13         tipo = row['type']
14
15         amountFrames = len(audioArray)
16         shiftAmount = np.random.randint(-int(amountFrames * maxShiftPct), int(
17             amountFrames * maxShiftPct))
18
19         shiftedAudioArray = timeShift(audioArray, shiftAmount)
20
21         dfNew = pd.DataFrame({'audio': [shiftedAudioArray], 'type': [tipo]})

```

```
21     dfAugmented = pd.concat([dfAugmented, dfNew], ignore_index=True)
22
23     dfResult = pd.concat([df, dfAugmented], ignore_index=True)
24
25     return dfResult
```

Listing 3.2: Aumento de datos para función de onda

### 3.4.2. Aumento de datos técnicas de aprendizaje profundo

Para el caso de la transformación de los audios en espectrogramas utilizados para entrenar la red neuronal no funcionaría aplicar técnicas como rotar los espectrogramas, desorganizarlos, entre otras técnicas o estrategias que causen una ruptura de la coherencia temporal del espectrograma. Por lo tanto, se utilizaron 3 técnicas para realizar el aumento de datos presentadas a continuación:

- **Desplazamiento Temporal:** Se mueve la señal de audio en el tiempo, simulando un cambio en el inicio de la grabación. Esto se realiza desplazando los datos de la señal de audio una cantidad aleatoria, lo que cambia el punto de inicio sin alterar la información frecuencial.
- **Desplazamiento de Frecuencia:** Se altera la frecuencia del audio, modificando la altura tonal sin cambiar la duración. Para ello, se aplica una Transformada de Fourier de Tiempo Corto (STFT) a la señal de audio, y luego se desplaza el espectrograma resultante verticalmente. La señal de audio desplazada se recupera mediante la Transformada Inversa de Fourier de Tiempo Corto (ISTFT).
- **Adición de Ruido:** Se agrega ruido blanco a la señal de audio, imitando condiciones de grabación menos ideales. Esto se realiza generando una señal de ruido aleatorio y sumándola a la señal de audio original, lo cual introduce pequeñas variaciones que pueden mejorar la robustez del modelo frente a ruido en los datos reales.

Finalmente, para cada audio se guarda la versión original del mismo y se elige una de las 3 técnicas aumento de forma aleatoria. Como resultado, se obtiene un dataset con 452 datos, duplicando la cantidad original de datos. El proceso incluye la carga del archivo de audio usando `librosa.load`, la generación del espectrograma mediante `librosa.stft`, y la conversión del espectrograma a una escala logarítmica con `librosa.amplitude_to_db`. Finalmente, el espectrograma se visualiza y se guarda como una imagen usando `librosa.display.specshow` y `plt.savefig` de `matplotlib`.

### 3.4.3. Conjuntos de entrenamiento y prueba.

#### 3.4.3.1. Técnicas clásicas.

Los conjuntos de entrenamiento y prueba fueron divididos utilizando el método de la librería `sklearn train_test_split`, dejando el 80 % de los datos para entrenamiento y el 20 % para prueba. Adicional, se hizo uso del parámetro del método mencionado `'stratify'` que permite garantizar la proporción de las clases en ambos conjuntos de datos, este parámetro es de vital importancia en

el presente trabajo, debido al desbalance de las clases, donde la aparición de una de las clases de forma mayoritaria en alguno de los dos conjuntos de datos puede llevar a conclusiones erróneas por el desbalance de las clases.

### 3.4.3.2. Técnicas de aprendizaje profundo.

En el caso de las técnicas de aprendizaje profundo el dataset fue dividido en un 70 % de los datos para entrenamiento y el 30 % sobrante se dividió en un 15 % para validación que es un conjunto de datos usado por el modelo durante el entrenamiento para realizar ajustes y probar el desempeño en cada época del modelo y el restante 15 % se destinó para el conjunto de prueba con el fin de ser usado únicamente cuando el entrenamiento finalice. Al igual que para las técnicas clásicas también se dividió el conjunto de datos de manera estratificada, conservando las proporciones entre las clases en cada división de datos para evitar que exista un desbalance entre las mismas al momento de evaluar y/o entrenar. Por lo tanto, en la tabla 3.4 se muestra un resumen de la cantidad de datos presentes en los tres conjuntos de entrenamiento para ambas clases.

Dataset	Tamaño	Norm (0)	Patho (1)
Training dataset	316	74	242
Validation dataset	68	16	52
Test dataset	68	16	52

Cuadro 3.4: Distribución de datos en los datasets

### 3.4.4. Visualización de las muestras.

Como menciona Gourisaria *et al.* [25], los audios pueden ser visualizados mediante espectrogramas, donde la frecuencia se muestra en el eje y en función del tiempo en el eje x. Otra forma de visualización es utilizando formas de onda, donde la amplitud se representa en el eje y en función del tiempo en el eje x.

En la figura 3.2 se muestra una representación de un espectrograma de una muestra sana, mientras que en la figura 3.3 se muestra un espectrograma de una muestra patológica. A su vez, en la figura 3.4 se muestra una gráfica de forma de onda de una muestra sana y en la figura 3.5 una muestra patológica.

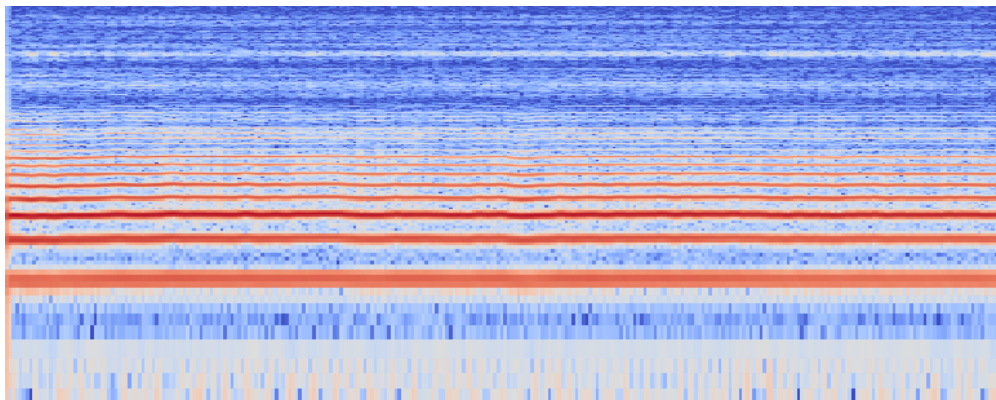


Figura 3.2: Espectrograma muestra saludable

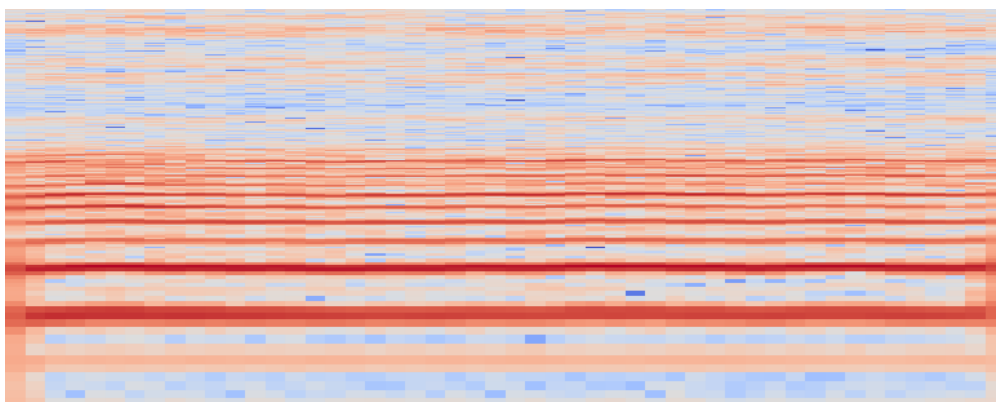


Figura 3.3: Espectrograma muestra patológica

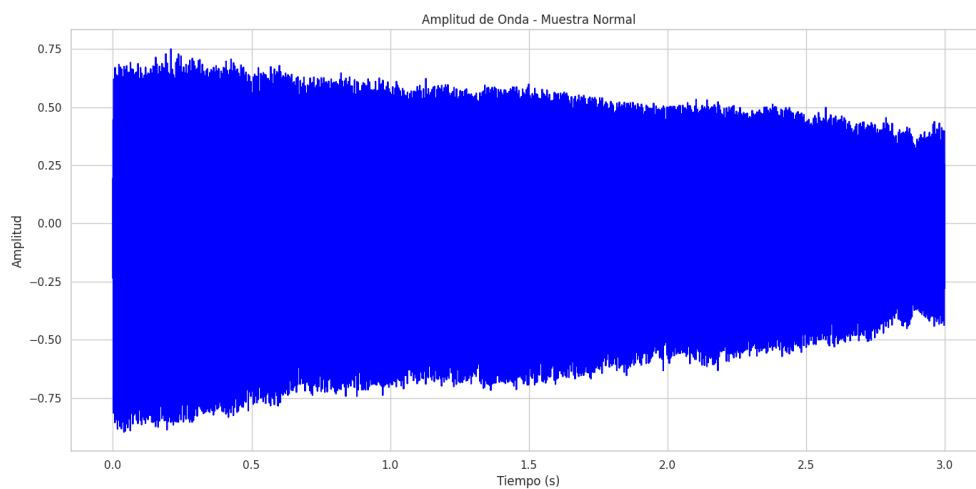


Figura 3.4: Amplitud de onda muestra normal

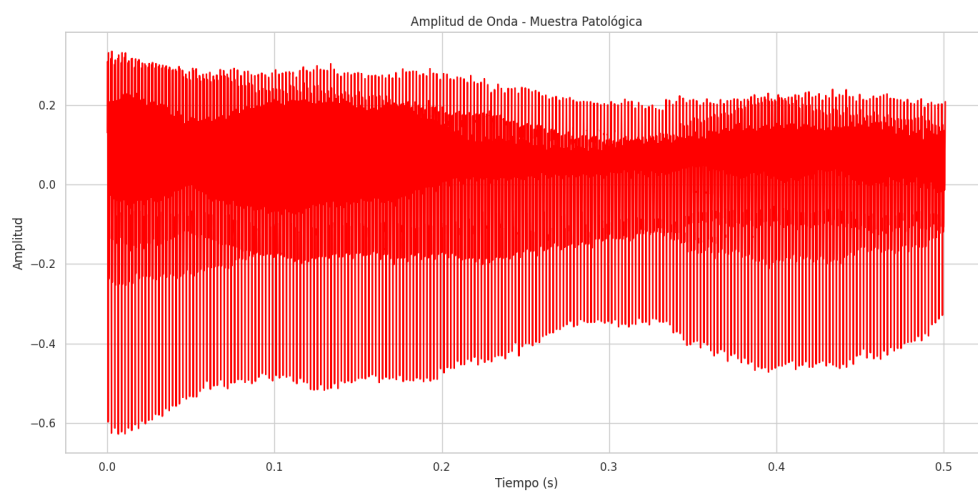


Figura 3.5: Amplitud de onda muestra patológica



# Desarrollo e implementación

---

En esta sección, se detalla el desarrollo e implementación de varios modelos de aprendizaje automático y profundo para la identificación de patologías en la voz, además del flujo de los datos a lo largo del proyecto.

## 4.1. Flujo de datos.

El flujo de datos es presentado en la imagen [4.1](#), la cual describe el flujo de los datos. Siendo la etapa inicial la obtención y lectura del dataset, posteriormente el preprocesamiento del conjunto de datos dividido en dos, una para las técnicas de aprendizaje clásicas y otra para las técnicas de aprendizaje profundo. Posteriormente, cuando se tienen los conjuntos de datos se procede a la etapa de entrenamiento y optimización de modelos, etapa que tiene como objetivo analizar de cada modelo escogido su sesgo de entrenamiento y su rendimiento inicial en la clasificación. Luego de ello, se pasa a la etapa de análisis de resultados, donde se analiza para cada modelo obtenido su desempeño principalmente en las métricas de f1-score y accuracy. Finalmente, cuando los resultados de todos los modelos son analizados y comparados se procede a desplegar los resultados de los modelos mediante una aplicación web en la cual se puedan cargar los audios, se procesen y permita conocer si la muestra es patológica o no lo es.

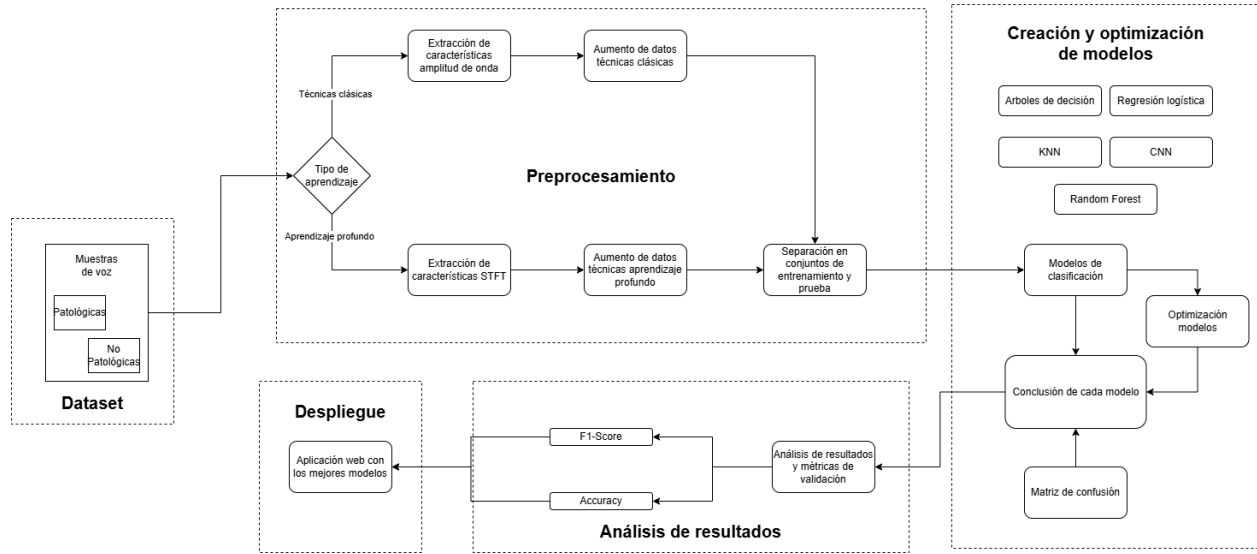


Figura 4.1: Etapas y flujo de datos

## 4.2. Técnicas clásicas de machine learning.

Cuando se mencionan técnicas clásicas de aprendizaje automático se hace referencia a todos los algoritmos que no involucran redes neuronales profundas. Por lo general, estos algoritmos han sido desarrollados e implementados desde los inicios del campo del aprendizaje automático. Estas técnicas incluyen algoritmos o métodos como regresión lineal, regresión logística, máquinas de soporte vectorial, K-Nearest Neighbors (KNN), árboles de decisión, random forest, naive bayes, entre otros algoritmos.

### 4.2.1. Entrenamiento

Para el entrenamiento de los modelos de aprendizaje automático se utilizó validación cruzada la cual es una técnica que permite evaluar el rendimiento de un modelo dividiendo el conjunto de datos en múltiples subconjuntos (pliegues). En cada iteración, el modelo se entrena en una combinación de pliegues y se evalúa en el pliegue restante, lo que proporciona una estimación más robusta y generalizable del rendimiento del modelo.

La implementación de esta técnica está definida en la librería ‘sklearn’ como ‘cross\_val\_score’ y recibe como parámetros el modelo a evaluar, los datos de características, las etiquetas, el número de pliegues (‘cv’), y la métrica de evaluación. Para el presente trabajo se hizo uso de un ‘cv’ de 5 y se tomó como métrica de validación el ‘f1-score’ definido como ‘f1\_macro’.

### 4.2.2. Evaluación de los modelos

La evaluación de los modelos se realizó utilizando técnicas de validación cruzada y reportes de clasificación. La validación cruzada se implementó mediante un esquema de k-folds estratificado, que asegura que cada pliegue contiene una proporción representativa de cada clase. Este método proporciona una estimación robusta del rendimiento del modelo al utilizar múltiples subconjuntos de datos para el entrenamiento y la validación, ayudando a prevenir el sobreajuste y asegurando que el modelo generalice bien a datos no vistos.

### 4.2.3. Optimización

La optimización de modelos de aprendizaje automático generalmente hace referencia a la búsqueda de hiperparámetros. Los hiperparámetros son parámetros del modelo que no se aprenden directamente a partir de los datos de entrenamiento, sino que se configuran antes del proceso de entrenamiento. Ejemplos de hiperparámetros incluyen la tasa de aprendizaje en redes neuronales, la profundidad máxima de un árbol de decisión, y el parámetro de regularización en modelos de regresión logística. No existe una fórmula o proceso que garantice encontrar los mejores hiperparámetros, por lo que la única opción se resume en probar el comportamiento de estos valores en los modelos.

La búsqueda de hiperparámetros puede realizarse de varias maneras, siendo las más comunes la búsqueda en cuadrícula (Grid Search) y la búsqueda aleatoria (Random Search). En la búsqueda en cuadrícula, se define un conjunto de posibles valores para cada hiperparámetro y se evalúan todas las combinaciones posibles. En la búsqueda aleatoria, se definen distribuciones de probabilidad para los hiperparámetros y se seleccionan combinaciones de valores de manera aleatoria.

Posterior a realizar la optimización, los modelos con los mejores hiperparámetros fueron entrenados nuevamente haciendo uso de validación cruzada para visualizar su rendimiento de forma más amplia.

### 4.2.4. Implementación Regresión logística

En el presente trabajo se utiliza el modelo de regresión logística tomando la implementación de la librería sklearn. Inicialmente, se realiza un entrenamiento utilizando validación cruzada sin especificar ningún hiperparámetro de forma explícita.

Posteriormente, se realiza la búsqueda aleatoria de hiperparámetros, en este caso los rangos de valores elegidos se presentan en la tabla 4.1.

Hiperparámetro	Valores
<i>penalty</i>	['l1', 'l2', 'elasticnet', 'none']
<i>C</i>	Valores en una escala logarítmica desde $10^{-4}$ hasta $10^4$
<i>solver</i>	['liblinear', 'saga', 'lbfgs']
<i>max_iter</i>	[1000, 2000, 3000]

Cuadro 4.1: Hiperparámetros utilizados en la búsqueda aleatoria

#### 4.2.5. Implementación K-Nearest Neighbors (KNN)

Para la implementación del algoritmo de KNN en el presente trabajo se hizo uso de la librería sklearn. El entrenamiento y validación se realizan de la misma forma que en la regresión logística.

Los hiperparámetros elegidos en la búsqueda aleatoria se presentan en la tabla 4.2.

Hiperparámetro	Valores
<i>n_neighbors</i>	Valores enteros del 1 al 29
<i>weights</i>	['uniform', 'distance']
<i>metric</i>	['euclidean', 'manhattan', 'chebyshev', 'minkowski']

Cuadro 4.2: Hiperparámetros utilizados en la búsqueda aleatoria para K-Nearest Neighbors

#### 4.2.6. Implementación árboles de decisión

Para la implementación de este método, nuevamente se hizo uso de la implementación de la librería sklearn al igual que en los modelos mencionados anteriormente.

La tabla 4.3 presenta los hiperparámetros usados en la búsqueda aleatoria.

Hiperparámetro	Valores
<i>max_depth</i>	[None] + Valores enteros del 1 al 19
<i>min_samples_split</i>	Valores enteros del 2 al 19
<i>min_samples_leaf</i>	Valores enteros del 1 al 19
<i>criterion</i>	['gini', 'entropy']
<i>splitter</i>	['best', 'random']

Cuadro 4.3: Hiperparámetros utilizados en la búsqueda aleatoria para Árboles de Decisión

#### 4.2.7. Implementación Random Forest

La tabla 4.4 presenta los hiperparámetros usados en la búsqueda aleatoria para el método de random forests.

Hiperparámetro	Valores
<i>n_estimators</i>	Valores enteros del 10 al 190 con incrementos de 10
<i>max_depth</i>	[None] + Valores enteros del 1 al 19
<i>min_samples_split</i>	Valores enteros del 2 al 19
<i>min_samples_leaf</i>	Valores enteros del 1 al 19
<i>bootstrap</i>	[True, False]
<i>criterion</i>	['gini', 'entropy']

Cuadro 4.4: Hiperparámetros utilizados en la búsqueda aleatoria para Random Forest

### 4.3. Técnicas de aprendizaje profundo

Cuando se mencionan técnicas de aprendizaje profundo, se hace referencia a aquellos algoritmos que utilizan redes neuronales de múltiples capas que modelan y aprenden representaciones complejas de los datos. Estos algoritmos han tenido impacto en el campo del aprendizaje automático debido a su fiabilidad y comportamiento en tareas como reconocimiento de imágenes, el procesamiento de lenguaje natural y análisis de la voz. Sin embargo, estos modelos requieren de un mayor costo computacional y el proceso para elegir las capas e hiperparámetros puede resultar mucho más arduo que en las técnicas clásicas.

Para la implementación de los modelos presentados a continuación se hizo uso de la librería Keras, la cual es una api de alto nivel de una de las librerías mas antiguas y robustas en Python para aprendizaje profundo, como lo es Tensorflow, la cual incluye implementaciones de muchas capas, como lo son las capas convolucionales, recurrentes o densas.

#### 4.3.1. Red Neuronal Convolucional (CNN)

Las Redes Neuronales Convolucionales (CNN) son adecuadas para tareas de procesamiento de señales debido a su capacidad para capturar características locales a través de sus capas de convolución. Sin embargo, es necesario recordar que en este caso la red neuronal será entrenada con el equivalente gráfico de la señal de voz, como lo es el espectrograma, en lugar de la señal de voz persé.

La red fue diseñada con múltiples capas convolucionales para extraer características del espectrograma de la señal de audio, seguidas de capas de pooling para reducir la dimensionalidad y capas densas para realizar la clasificación. La función de pérdida utilizada fue la entropía cruzada y el optimizador elegido fue *Adam*.

El diseño de la red incluyó:

- **Capas Convolucionales:** Para la extracción de características locales, se utilizaron varias capas convolucionales con diferentes tamaños de filtros.
- **Capas de Pooling:** Para la reducción de la dimensionalidad y la generalización de las características extraídas.

- **Capas Densas:** Para la clasificación final, utilizando funciones de activación como ReLU y Softmax.

#### 4.3.1.1. Entrenamiento de la CNN

El modelo fue entrenado utilizando un conjunto de datos dividido en entrenamiento y validación. Se realizaron múltiples épocas de entrenamiento para asegurar la convergencia del modelo, y se monitorizaron métricas como la pérdida y la precisión para evaluar su rendimiento.

```
1 model = Sequential([
2     Conv2D(32, (3, 3), activation='relu', input_shape=(images.shape[1], images.
3         shape[2], images.shape[3])),
4     MaxPooling2D((2, 2)),
5     Conv2D(64, (3, 3), activation='relu'),
6     MaxPooling2D((2, 2)),
7     Conv2D(128, (3, 3), activation='relu'),
8     MaxPooling2D((2, 2)),
9     Flatten(),
10    Dense(128, activation='relu'),
11    Dropout(0.5),
12    Dense(1, activation='sigmoid')
13 ])
14 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Listing 4.1: Arquitectura de la CNN

Este código ilustra la arquitectura de la CNN utilizada y los pasos para su compilación. La figura 4.2 muestra la arquitectura de la red diseñada.

#### 4.3.2. Entrenamiento

Para el entrenamiento del modelo se utilizó la función 'fit' que hace parte de la implementación de Keras para los modelos de aprendizaje profundo. Se definieron 10 épocas de entrenamiento, con un tamaño de bloque de 32. Se guardan los resultados del entrenamiento para ser graficados y explorados en futuros capítulos.

##### 4.3.2.1. Optimización del modelo

La optimización del modelo hizo uso principalmente de Keras Tuner para optimizar los hiperparámetros de un modelo de red neuronal convolucional (CNN). El proceso consiste en definición de una función `build_model`, la cual define un modelo de CNN. Esta función recibe como parámetro los hiperparámetros que serán optimizados. En este caso, el número de filtros, el tamaño del kernel en las capas convolucionales, el número de unidades en la capa densa, la tasa de dropout y la tasa de aprendizaje del optimizador.

El modelo se construye agregando varias capas convolucionales y de pooling. En cada capa convolucional, el número de filtros y el tamaño del kernel se especifican como hiperparámetros a

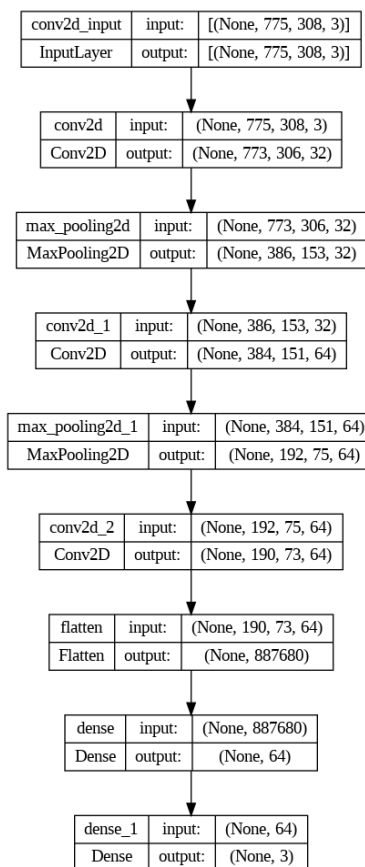


Figura 4.2: Arquitectura red neuronal CNN

optimizar. Posteriormente, se añaden capas densas y de dropout con hiperparámetros específicos. Finalmente, se compila el modelo utilizando el optimizador Adam, donde la tasa de aprendizaje es otro hiperparámetro a optimizar.

Para realizar la optimización de los hiperparámetros, se crea un objeto RandomSearch de Keras Tuner. Este objeto configura la búsqueda aleatoria de los mejores hiperparámetros para el modelo, definiendo el número máximo de combinaciones de hiperparámetros a probar (`max_trials`) y el número de ejecuciones por cada combinación (`executions_per_trial`). La búsqueda se inicia entrenando el modelo con los datos de entrenamiento y validación, con 10 épocas de entrenamiento.

Una vez finalizada la búsqueda, se obtienen los mejores hiperparámetros encontrados. Estos hiperparámetros se utilizan para construir y entrenar nuevamente el modelo con el fin de obtener un modelo final optimizado.

El bloque de código mostrado a continuación permite visualizar la búsqueda de hiperparámetros realizada para este modelo.

```

1 def build_model(hp):
2     model = Sequential()

```

```

3     model.add(Conv2D(
4         filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
5         kernel_size=hp.Choice('conv_1_kernel', values=[3,5]),
6         activation='relu',
7         input_shape=(images.shape[1], images.shape[2], images.shape[3])
8     ))
9     model.add(MaxPooling2D(pool_size=(2, 2)))
10
11    model.add(Conv2D(
12        filters=hp.Int('conv_2_filter', min_value=32, max_value=128, step=16),
13        kernel_size=hp.Choice('conv_2_kernel', values=[3,5]),
14        activation='relu'
15    ))
16    model.add(MaxPooling2D(pool_size=(2, 2)))
17
18    model.add(Conv2D(
19        filters=hp.Int('conv_3_filter', min_value=32, max_value=128, step=16),
20        kernel_size=hp.Choice('conv_3_kernel', values=[3,5]),
21        activation='relu'
22    ))
23    model.add(MaxPooling2D(pool_size=(2, 2)))
24
25    model.add(Flatten())
26    model.add(Dense(
27        units=hp.Int('dense_units', min_value=32, max_value=128, step=16),
28        activation='relu'
29    ))
30    model.add(Dropout(hp.Float('dropout_rate', min_value=0.2, max_value=0.5, step
31    =0.1)))
32    model.add(Dense(1, activation='sigmoid'))
33
34    model.compile(
35        optimizer=Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])),
36        loss='binary_crossentropy',
37        metrics=['accuracy']
38    )
39
40    return model
41
42    tuner = kt.RandomSearch(
43        build_model,
44        objective='val_accuracy',
45        max_trials=5,
46        executions_per_trial=3,
47        directory='Modelo Optimizado',
48        project_name='tesisModel'
49    )
50    tuner.search(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
51

```



```
52 best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
```

Listing 4.2: Optimización CNN

# Resultados

---

El presente capítulo tiene como objetivo mostrar los resultados de los modelos entrenados previamente, así como también los mejores parámetros encontrados, matrices de confusión y los resultados del entrenamiento por validación cruzada. También, el presente capítulo mostrará gráficas alucivas al entrenamiento, detallará las métricas obtenidas por cada modelo y brindará una comparación entre cada uno de ellos en cuestión de tiempo de entrenamiento y resultados.

## 5.1. Regresión logística

### 5.1.1. Modelo base

La regresión logística planteada inicialmente sin ninguna selección de hiperparámetros dio como resultado un `f1_score` promedio de 0.47 en el conjunto de entrenamiento, con unos puntajes individuales dentro de la validación cruzada de 0.493, 0.492, 0.433, 0.433 y 0.492. El `f1_score` del modelo en el conjunto de prueba fue de 0.66. Sin embargo, los resultados obtenidos con este modelo parecen indicar que la relación entre los datos no está dada por una función lineal.

La matriz de confusión que obtuvo la regresión lineal sin optimización se presenta en la tabla 5.2, estos resultados indican que el modelo sin optimización tiene un fuerte sesgo hacia la Clase 1, clasificando todas las instancias como pertenecientes a esta. Esto resulta en una alta precisión para la Clase 1, pero una total incapacidad para identificar correctamente la Clase 0, con 21 falsos positivos y ningún verdadero negativo.

El reporte de clasificación para la regresión logística sin optimización se presenta en la tabla 5.1. Los resultados obtenidos en el reporte de clasificación indican que el modelo sin optimización tiene un rendimiento desequilibrado. La alta precisión y el `recall` para la Clase 1 indican que el modelo identifica correctamente la mayoría de las instancias de esta clase. Sin embargo, la precisión y el `recall` para la Clase 0 son nulos, lo que muestra que el modelo no identifica correctamente ninguna instancia de esta clase. Esto sugiere que el modelo sin optimización está sesgado hacia la Clase 1 y tiene dificultades para reconocer la Clase 0.

Clase	Precisión	Recall	F1-Score	Soporte
0	0.00	0.00	0.00	21
1	0.77	1.00	0.87	70
<b>Exactitud</b>			0.77	91
<b>Promedio Macro</b>	0.38	0.50	0.43	91
<b>Promedio Ponderado</b>	0.59	0.77	0.67	91

Cuadro 5.1: Reporte de clasificación para la regresión logística sin optimización

Predicción\Valor Real	Clase 0	Clase 1
<b>Clase 0</b>	0	21
<b>Clase 1</b>	0	70

Cuadro 5.2: Matriz de confusión para la regresión logística sin optimización

Cabe resaltar que los datos suministrados a la regresión lineal para su entrenamiento están normalizados en una escala de  $[0, 1]$ , ya que este modelo puede resultar sensible a datos no normalizados.

### 5.1.2. Modelo optimizado

Los mejores hiperparámetros encontrados para este modelo son presentados en la tabla 5.3. El `f1_score` en entrenamiento del modelo optimizado usando validación cruzada es de 0.524 y los resultados individuales de cada uno son 0.5788, 0.494, 0.389, 0.566 y 0.589, estos resultados plantean una mejoría con respecto al modelo sin optimización de aproximadamente un 10 %, pasando de 0.47 a 0.524. En prueba, el modelo obtuvo un `f1_score` de 0.69, presentando una mejora de 0.03 con respecto al modelo sin optimización. La matriz de confusión del modelo, presentada en la tabla 5.4, sugiere que el modelo tiene una alta capacidad para identificar correctamente la Clase 1, con 61 verdaderos positivos. Sin embargo, hay un número significativo de falsos positivos (17), lo que indica que en algunos casos el modelo clasifica incorrectamente las instancias de la Clase 0 como Clase 1.

A su vez, el reporte de clasificación para el modelo con optimización 5.5. Estos resultados indican que la exactitud general del modelo es del 71 %, esto apunta a una mejora en la capacidad del modelo para identificar correctamente ambas clases. Los promedios macro y ponderado también reflejan una mejor distribución del rendimiento del modelo entre las clases, lo que sugiere que la optimización de los hiperparámetros ha llevado a un modelo más equilibrado y eficaz. Sin embargo, las métricas generales del modelo parecen indicar que no es el adecuado para esta tarea debido a su sesgo.

Hiperparámetro	Valor
<i>solver</i>	saga
<i>penalty</i>	l1
<i>max_iter</i>	1000
<i>C</i>	0.1

Cuadro 5.3: Mejores Hiperparámetros para Regresión Logística

Predicción\Valor Real	Clase 0	Clase 1
Clase 0	4	17
Clase 1	9	61

Cuadro 5.4: Matriz de confusión para la regresión logística con optimización

Clase	Precisión	Recall	F1-Score	Soporte
0	0.31	0.19	0.24	21
1	0.78	0.87	0.82	70
<b>Exactitud</b>			0.71	91
<b>Promedio Macro</b>	0.54	0.53	0.53	91
<b>Promedio Ponderado</b>	0.67	0.71	0.69	91

Cuadro 5.5: Reporte de clasificación para la regresión logística con optimización

## 5.2. KNN

### 5.2.1. Modelo base

Inicialmente el entrenamiento sin optimización y con los parámetros default del modelo entrenado usando validación cruzada dio como resultado un promedio de un `f1_score` en entrenamiento de 0.433 con unos puntajes individuales en la validación cruzada de 0.434, 0.433, 0.433, 0.433 y 0.433. En el conjunto de prueba el modelo obtuvo un `f1_score` de 0.668.

La matriz de confusión se presenta en la tabla 5.6, al igual que el reporte de clasificación se presenta en la tabla 5.7. Ambas métricas muestran un rendimiento muy desigual entre las dos clases. Para la clase 0, el modelo presenta un `f1-score` de 0.00, lo que indica que no hubo ninguna predicción correcta para esta clase. Esto significa que el modelo no identificó correctamente ninguna instancia de la clase 0, resultando en una precisión y recall de 0.00 para esta clase.

En contraste, para la clase 1, el modelo muestra un rendimiento significativamente mejor. La precisión es de 0.77, y el recall es de 1.00, lo que significa que el modelo identifica correctamente

todas las instancias de la clase 1. El f1-score para la clase 1 es de 0.87, sugiriendo un buen equilibrio entre precisión y recall para esta clase. La exactitud general del modelo es de 0.77, pero este valor es sesgado por el buen rendimiento en la clase 1 y el fallo completo en la clase 0.

Predicción\Valor Real	Clase 0	Clase 1
Clase 0	0	21
Clase 1	0	70

Cuadro 5.6: Matriz de confusión para KNN sin optimización

Clase	Precisión	Recall	F1-Score	Soporte
0	0.00	0.00	0.00	21
1	0.77	1.00	0.87	70
<b>Exactitud</b>			0.77	91
<b>Promedio Macro</b>	0.38	0.50	0.43	91
<b>Promedio Ponderado</b>	0.59	0.77	0.67	91

Cuadro 5.7: Reporte de clasificación para KNN sin optimización

### 5.2.2. Modelo optimizado

Los mejores hiperparámetros encontrados por la búsqueda aleatoria se presentan en la tabla 5.8. Sin embargo, estos resultados parecen tampoco mejorar en demasía el modelo original, dejando nuevamente como resultado un f1\_score de 0.619 en entrenamiento, con unos resultados individuales de 0.680, 0.571, 0.576, 0.618 y 0.653. En el conjunto de prueba el f1\_score obtenido por el modelo es de 0.688.

La matriz de confusión se presenta en la tabla 5.10 y el reporte de clasificación en la tabla 5.9. Ambas métricas indican que el sesgo del modelo pareciera no ser adecuado para esta tarea de clasificación, sin embargo, el rendimiento del modelo KNN optimizado en comparación con el modelo sin optimizar fue superior, el modelo optimizado muestra un f1-score de 0.24 para la clase 0 y 0.82 para la clase 1, con una precisión general de 0.71. En contraste, el modelo sin optimización tiene un f1-score de 0.00 para la clase 0 y 0.87 para la clase 1, con una precisión general de 0.77. Aunque el modelo sin optimización muestra un excelente rendimiento para la clase 1, no clasifica de forma correcta la clase 0, lo que hace al modelo optimizado un modelo más equilibrado entre ambas clases.

Hiperparámetro	Valor
<i>weights</i>	uniform
<i>n_neighbors</i>	2
<i>metric</i>	chebyshev

Cuadro 5.8: Mejores Hiperparámetros para KNN

Clase	Precisión	Recall	F1-Score	Soporte
0	0.31	0.19	0.24	21
1	0.78	0.87	0.82	70
<b>Exactitud</b>			0.71	91
<b>Promedio Macro</b>	0.54	0.53	0.53	91
<b>Promedio Ponderado</b>	0.67	0.71	0.69	91

Cuadro 5.9: Reporte de clasificación para KNN con optimización

Predicción\Valor Real	Clase 0	Clase 1
<b>Clase 0</b>	4	17
<b>Clase 1</b>	9	61

Cuadro 5.10: Matriz de confusión para KNN con optimización

## 5.3. Árboles de decisión

### 5.3.1. Modelo base

En el entrenamiento inicial de los árboles de decisión, los resultados fueron prometedores, dando como resultado un `f1_score` en entrenamiento promedio de 0.846 con resultados individuales equivalentes a 0.902, 0.941, 0.762, 0.794 y 0.831. El `f1_score` del modelo en el conjunto de prueba fue de 0.9. La matriz de confusión es presentada en la tabla 5.11 y los resultados indican que el modelo de árbol de decisión sin optimización tiene una alta capacidad para identificar correctamente las instancias de la Clase 1, con una tasa muy baja de falsos negativos (solo 2). También muestra un rendimiento razonable para la Clase 0, con una precisión alta y 7 falsos positivos.

El reporte de clasificación 5.12 indican que el modelo de árbol de decisión sin optimización tiene un rendimiento muy fuerte en la Clase 1, con un buen equilibrio entre precisión y recall, y con un `f1_score` en el conjunto de prueba de 0.9.

Predicción\Valor Real	Clase 0	Clase 1
Clase 0	14	7
Clase 1	2	68

Cuadro 5.11: Matriz de confusión para el árbol de decisión sin optimización

Clase	Precisión	Recall	F1-Score	Soporte
0	0.88	0.67	0.76	21
1	0.91	0.97	0.94	70
<b>Exactitud</b>			0.90	91
<b>Promedio Macro</b>	0.89	0.82	0.85	91
<b>Promedio Ponderado</b>	0.90	0.90	0.90	91

Cuadro 5.12: Reporte de clasificación para el árbol de decisión sin optimización

### 5.3.2. Modelo optimizado

Los mejores hiperparámetros del modelo son definidos en la tabla 5.13. El `f1_score` obtenido en entrenamiento después de realizar la optimización es de 0.9, con unos puntajes individuales de 0.806, 0.922, 0.884, 0.943 y 0.943. En el conjunto de prueba el `f1_score` obtenido por el modelo fue de 0.830.

El reporte de clasificación 5.14 y la matriz de confusión 5.15, indican que el modelo de árbol de decisión sin optimización superó al optimizado en todas las métricas clave. En términos de precisión, recall, y f1-score, el modelo sin optimización mostró un rendimiento superior tanto para la clase 0 como para la clase 1. Específicamente, el modelo sin optimización tuvo una precisión más alta al predecir ambas clases, logrando un recall del 97 % para la clase 1 frente al 91 % del modelo optimizado. Además, el `f1_score` fue consistentemente mejor para el modelo sin optimización, alcanzando 0.94 para la clase 1 en comparación con 0.90 del optimizado. La exactitud general del modelo sin optimización fue del 90 %, mientras que el modelo optimizado solo logró un 84 %. Estas diferencias se reflejaron también en los promedios macro y ponderados del f1-score, donde el modelo sin optimización tuvo un rendimiento notablemente mejor. Por lo tanto, a pesar de los esfuerzos de optimización, el modelo sin optimización demostró ser más efectivo en la clasificación de las instancias en el conjunto de prueba.

Hiperparámetro	Valor
<i>splitter</i>	best
<i>min_samples_split</i>	3
<i>min_samples_leaf</i>	19
<i>max_depth</i>	7
<i>criterion</i>	entropy

Cuadro 5.13: Mejores Hiperparámetros para Árbol de Decisión

Clase	Precisión	Recall	F1-Score	Soporte
0	0.67	0.57	0.62	21
1	0.88	0.91	0.90	70
<b>Exactitud</b>			0.84	91
<b>Promedio Macro</b>	0.77	0.74	0.76	91
<b>Promedio Ponderado</b>	0.83	0.84	0.83	91

Cuadro 5.14: Reporte de clasificación para el árbol de decisión con optimización

Predicción\Valor Real	Clase 0	Clase 1
<b>Clase 0</b>	12	9
<b>Clase 1</b>	6	64

Cuadro 5.15: Matriz de confusión para el árbol de decisión con optimización

## 5.4. Random forest

### 5.4.1. Modelo base

En el caso del modelo de random forest, sufre un fenómeno similar al de los árboles de decisión, algo que tiene congruencia dado que random forest es un modelo que ensambla cierta cantidad de árboles de decisión. En el entrenamiento inicial, sin optimización el modelo arrojó como `f1_score` en entrenamiento un valor promedio de 0.95, con unos puntajes individuales de 0.963, 0.922, 0.943, 0.981 y 0.962, siendo el mayor valor obtenido por algún modelo sin realizar ninguna optimización en el conjunto de entrenamiento. En el conjunto de prueba el modelo obtuvo un `f1_score` de 0.917. En la matriz de confusión 5.16 se tienen 70 verdaderos positivos y 0 falsos positivos para la clase 1, en el caso de la clase 0, se tienen 14 verdaderos negativos y 7 falsos positivos. Por consiguiente, este modelo ha logrado identificar perfectamente la clase 1 en el conjunto de prueba, sin embargo, el rango de mejora para la clase 0 sigue siendo alto.



El reporte de clasificación 5.17 reafirma los resultados proporcionados por la matriz de confusión en los cuales el modelo se comporta perfecto en el conjunto de entrenamiento con la clase 1 con un `f1_score` de 0.95, y para la clase 0 el modelo ha logrado un `f1_score` de 0.80.

Predicción\Valor Real	Clase 0	Clase 1
Clase 0	14	7
Clase 1	0	70

Cuadro 5.16: Matriz de confusión para el árbol de decisión con optimización

Clase	Precisión	Recall	F1-Score	Soporte
0	1.00	0.67	0.80	21
1	0.91	1.00	0.95	70
<b>Exactitud</b>			0.92	91
<b>Promedio Macro</b>	0.95	0.83	0.88	91
<b>Promedio Ponderado</b>	0.93	0.92	0.92	91

Cuadro 5.17: Reporte de clasificación para Random Forest sin optimización

#### 5.4.2. Modelo optimizado

Al realizar la búsqueda de hiperparámetros, cuyos resultados son mostrados en la tabla 5.19 y optimizar el modelo, el valor promedio de `f1_score` en entrenamiento fue de 0.966, con unos puntajes individuales 0.963, 0.943, 0.981, 0.981 y 0.962. La matriz de confusión es mostrada en la tabla 5.18. El valor del `f1_score` en el conjunto de prueba fue de 0.917, el mismo que el modelo sin optimización, al igual que la matriz de confusión. En el reporte de clasificación 5.20 el modelo de random forest también obtuvo los mismos resultados que el modelo sin optimización.

En conclusión, pese a realizar la búsqueda de hiperparámetros e intentar optimizar el modelo, los hiperparámetros del modelo por defecto parecen tener los mismos resultados en el conjunto de prueba que los encontrados en la búsqueda. Sin embargo, a pesar de todo ello, el modelo de random forest es el que ha obtenido un mayor desempeño en cuanto a las técnicas clásicas de aprendizaje automático.

Predicción\Valor Real	Clase 0	Clase 1
Clase 0	14	7
Clase 1	0	70

Cuadro 5.18: Matriz de confusión para Random Forest sin optimización

Hiperparámetro	Valor
<i>n_estimators</i>	30
<i>min_samples_split</i>	19
<i>min_samples_leaf</i>	1
<i>max_depth</i>	12
<i>criterion</i>	entropy
<i>bootstrap</i>	True

Cuadro 5.19: Mejores Hiperparámetros para Random Forest

Clase	Precisión	Recall	F1-Score	Soporte
0	1.00	0.67	0.80	21
1	0.91	1.00	0.95	70
<b>Exactitud</b>			0.92	91
<b>Promedio Macro</b>	0.95	0.83	0.88	91
<b>Promedio Ponderado</b>	0.93	0.92	0.92	91

Cuadro 5.20: Reporte de clasificación para Random Forest con optimización

El tiempo que tomó el modelo en realizar la búsqueda de hiperparámetros fue de minutos.

## 5.5. Red Neuronal Convulucional

### 5.5.1. Modelo base

En este caso, el modelo de CNN sin optimización durante el entrenamiento alcanzó en la última época de entrenamiento un accuracy de 0.9671, al igual que un accuracy en el conjunto de validación durante el entrenamiento de 0.9412. Con un valor de loss de 0.0738 en el conjunto de entrenamiento y 0.1261 en el conjunto de validación. La gráfica del entrenamiento de la función perdida y el accuracy es presentada en la figura 5.1 la cual indica que el modelo no presenta sobreajuste, maldición de la dimensión o gradientes que se comportan de manera no deseada (explotan o se desvanecen), ya que la pérdida en entrenamiento disminuye al mismo tiempo que el loss en validación, al igual que el accuracy aumenta tanto en entrenamiento como en validación con el paso de las épocas.

En el conjunto de prueba el modelo obtuvo como resultado un f1-score de 1.0 y un accuracy de 1.0. La matriz de confusión obtenida por el modelo es presentada en 5.21, al igual que el reporte de clasificación es presentado en 5.22. Ambos confirman que los resultados del modelo en el conjunto de prueba son perfectos y el clasificador ha conseguido identificar de manera correcta los 68 datos presentes en este conjunto, sin embargo, esto no necesariamente implica que el modelo sea perfecto y logre clasificar todos los audios referentes a patologías en el conjunto universal de audios.

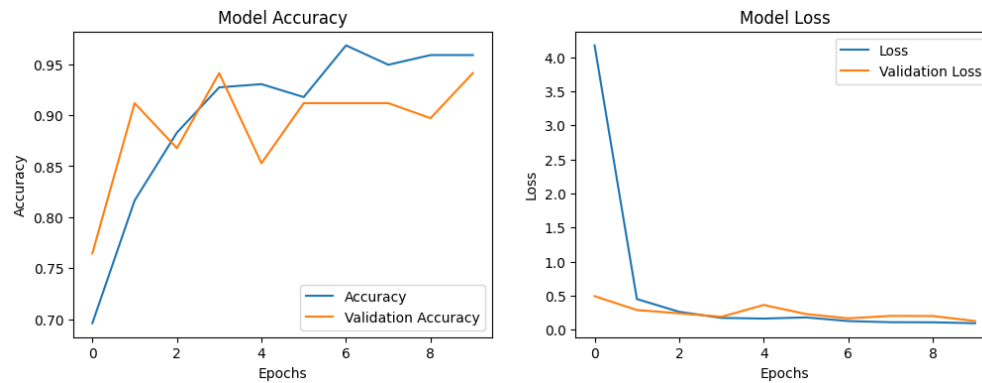


Figura 5.1: Entrenamiento CNN sin optimización

Predicción\Valor Real	Clase 0	Clase 1
Clase 0	16	0
Clase 1	0	52

Cuadro 5.21: Matriz de confusión para CNN sin optimización

	Precision	Recall	F1-score	Support
<b>Norm</b>	1.00	1.00	1.00	16
<b>Patho</b>	1.00	1.00	1.00	52
<b>Accuracy</b>			1.00	68
<b>Macro avg</b>	1.00	1.00	1.00	68
<b>Weighted avg</b>	1.00	1.00	1.00	68

Cuadro 5.22: Reporte de clasificación

### 5.5.2. Modelo optimizado

El tiempo requerido por la búsqueda aleatoria para encontrar el mejor modelo fue de 1h 18m 17s. La tabla 5.23 presenta las capas e hiperparámetros del mejor modelo encontrado. Sin embargo a continuación se presentan algunos resultados también obtenidos por esta búsqueda que no se reflejan en la capas del modelo:

- El mejor número de filtros en la primera capa convolucional es 48.
- El mejor tamaño de kernel en la primera capa convolucional es 3.
- El mejor número de filtros en la segunda capa convolucional es 80.
- El mejor tamaño de kernel en la segunda capa convolucional es 3.
- El mejor número de filtros en la tercera capa convolucional es 128.

- El mejor tamaño de kernel en la tercera capa convolucional es 5.
- El mejor número de unidades en la capa densa es 128.
- La mejor tasa de dropout es 0.30000000000000004.
- El mejor learning rate es 0.0001.

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 306, 773, 32)	1,184
max_pooling2d_3 (MaxPooling2D)	(None, 153, 386, 32)	0
conv2d_4 (Conv2D)	(None, 151, 384, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 75, 192, 64)	0
conv2d_5 (Conv2D)	(None, 73, 190, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 36, 95, 128)	0
flatten_1 (Flatten)	(None, 437760)	0
dense_2 (Dense)	(None, 128)	56,033,408
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

Cuadro 5.23: Características del mejor modelo encontrado por la búsqueda aleatoria

Los resultados en entrenamiento parecen ser similares a los resultados del modelo base, alcanzan-  
do el un accuracy de 0.9672 durante el entrenamiento, con un accuracy de 0.9706 en el conjunto de  
validación. La gráfica de accuracy y loss durante el entrenamiento es presentada en 5.2 y demuestra  
que en las 10 épocas de entrenamiento el modelo aumentó su accuracy y disminuyó el loss en los  
conjuntos de validación y entrenamiento. Finalmente, en el conjunto de prueba el modelo obtuvo  
un 0.97 de f1-score. La matriz de confusión presentada en la tabla 5.24 muestra que el modelo ha  
logrado identificar perfectamente las muestras de la clase patológica, sin embargo, para las muestras  
de la clase no patológica de las 16 muestras 2 fueron clasificadas incorrectamente por el modelo.

El reporte de entrenamiento del modelo es presentado en la tabla 5.25 y muestra que para la  
clase "Norm" se obtuvo una precisión de 1.00, un recall de 0.88 y un F1-score de 0.93 con un soporte  
de 16 instancias, mientras que para la clase "Patho" se obtuvo una precisión de 0.96, un recall de  
1.00 y un F1-score de 0.98 con un soporte de 52 instancias. La precisión global del modelo fue de  
0.97 sobre 68 instancias de prueba. La media macro (macro avg) de precisión, recall y F1-score  
fueron 0.98, 0.94 y 0.96 respectivamente, y la media ponderada (weighted avg) de precisión, recall  
y F1-score fueron todas 0.97.

Predicción\Valor Real	Clase 0	Clase 1
Clase 0	14	2
Clase 1	0	52

Cuadro 5.24: Matriz de confusión para CNN optimizada

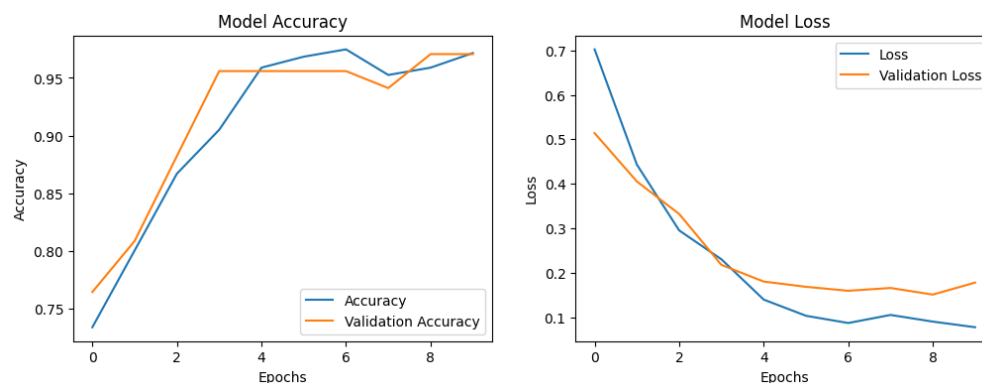


Figura 5.2: Entrenamiento CNN optimizada

	Precision	Recall	F1-score	Support
<b>Norm</b>	1.00	0.88	0.93	16
<b>Patho</b>	0.96	1.00	0.98	52
<b>Accuracy</b>			0.97	68
<b>Macro avg</b>	0.98	0.94	0.96	68
<b>Weighted avg</b>	0.97	0.97	0.97	68

Cuadro 5.25: Reporte de entrenamiento de la red neuronal CNN optimizada

## 5.6. Comparativa de modelos

Cabe resaltar que los valores de f1-score en entrenamiento para los modelos de aprendizaje profundo fueron omitidos, ya que durante este proceso la métrica utilizada fue el accuracy para estos modelos, por lo tanto, no se puede realizar una comparativa en igualdad de conceptos utilizando esta métrica.

Los resultados de los modelos se presentan en la tabla 5.26, utilizando el f1\_score como la principal métrica de evaluación. De las técnicas clásicas de aprendizaje automático, el modelo con el mejor desempeño en el conjunto de prueba es el modelo de random\_forest, obteniendo un f1-score de 0.917 en este conjunto. Sin embargo, para el conjunto de prueba los modelos de aprendizaje profundo superaron en un 6 % al modelo de random forest, consiguiendo así el modelo de CNN un f1-score de 0.97 en el conjunto de prueba.

El modelo con el peor desempeño en entrenamiento fue el modelo de KNN sin optimización, obteniendo un f1\_score de 0.433, siendo este incluso inferior al de regresión logística. A su vez, en prueba hay 2 modelos con el mismo f1\_score, siendo estos la regresión logística sin optimización y KNN sin optimización, ambos con un f1\_score de 0.668.

Modelo	Optimización	F1-Score Training	F1-Score Test
<b>Logistic Regression</b>	Sin optimización	0.468	0.668
<b>Logistic Regression</b>	Con optimización	0.524	0.694
<b>KNN</b>	Sin optimización	0.433	0.668
<b>KNN</b>	Con optimización	0.619	0.688
<b>Decision Tree</b>	Sin optimización	0.846	0.896
<b>Decision Tree</b>	Con optimización	0.900	0.830
<b>Random Forest</b>	Sin optimización	0.954	0.917
<b>Random Forest</b>	Con optimización	0.966	0.917
<b>CNN</b>	Sin optimización		0.961
<b>CNN</b>	Con optimización		0.97

Cuadro 5.26: Comparación de F1-Scores de Modelos

### 5.6.1. Mejor modelo

Las mejores métricas las obtuvo el modelo de CNN, el cual logró mejorar el resultado obtenido del mejor modelo de las técnicas clásicas en casi un 6 %, pasando de un 0.917 de f1-Score a un 0.97 de f1-score.

# Despliegue

---

## 6.1. Despliegue de la Aplicación de Detección de Voz Patológica

El despliegue de la aplicación de detección de voz patológica se realizó utilizando **FastAPI**, un framework moderno y de alto rendimiento basado en Python, ideal para construir API's con un enfoque en la velocidad y simplicidad. Además, se utilizaron plantillas HTML para la interfaz de usuario, las cuales fueron estilizadas con CSS para proporcionar una experiencia visual atractiva y moderna.

### 6.1.1. Estructura del Proyecto

El proyecto se organizó en diferentes directorios y archivos para mantener una estructura clara y modular. Los archivos principales son:

- **main.py**: Contiene el código principal de la aplicación, incluyendo la definición de rutas, carga de modelos, y la lógica para realizar predicciones.
- **templates/**: Directorio que contiene los archivos HTML, que son utilizados para renderizar la interfaz de usuario.
- **static/**: Directorio que almacena los archivos CSS y otros recursos estáticos como imágenes.
- **uploads/**: Directorio donde se guardan los espectrogramas generados a partir de los archivos de audio.
- **modelo\_random\_forest.pkl** y **modeloOptimizadoCNN.h5**: Archivos que contienen los modelos entrenados para realizar las predicciones.

### 6.1.2. Carga y Uso de Modelos de Machine Learning

Para la predicción, se utilizaron dos modelos de Machine Learning diferentes:

- **Random Forest**: Modelo entrenado para detectar patrones en datos de audio y clasificar las voces como patológicas o normales. Se utiliza la biblioteca **pickle** para cargar este modelo.
- **CNN (Red Neuronal Convolucional)**: Modelo de aprendizaje profundo optimizado para trabajar con imágenes de espectrogramas. Este modelo fue cargado utilizando la biblioteca **TensorFlow**.

### 6.1.3. Interfaz de Usuario y Estilización

La interfaz de usuario se diseñó utilizando HTML y se estilizó con CSS, almacenado en el directorio `static/`. Se proporcionó una página principal donde el usuario puede seleccionar el modelo a utilizar y cargar un archivo de audio en formato `.wav`. Al enviar el archivo, la aplicación procesa la solicitud y redirige al usuario a una nueva página donde se muestran los resultados de la predicción.

### 6.1.4. Procesamiento de Archivos de Audio

El procesamiento de los archivos de audio depende del modelo seleccionado:

- Para el modelo de **Random Forest**, se lee el archivo de audio, se normaliza y se extraen las características necesarias para realizar la predicción.
- Para el modelo **CNN**, se genera un espectrograma a partir del archivo de audio, el cual es convertido en una imagen que es utilizada como entrada para el modelo.

### 6.1.5. Despliegue y Funcionamiento

La aplicación está diseñada para funcionar como un servicio web accesible a través de un navegador. Utilizando **FastAPI**, se definieron rutas para la página principal (`/`) y para manejar las predicciones (`/predict`). Al recibir un archivo de audio y el tipo de modelo a utilizar, la aplicación procesa el archivo, realiza la predicción y muestra los resultados en una nueva página. El uso de **FastAPI** y **Python** permitió un despliegue eficiente y rápido de la aplicación, manteniendo la simplicidad en la implementación y ofreciendo una experiencia de usuario fluida. La gráfica 6.1 presenta la interfaz inicial, la cual permite cargar un archivo de audio. La 6.2 presenta el resultado después de haber enviado el archivo de audio, en este caso, muestra que la muestra pertenece a la clase patológica y presenta una patología con casi un 1 % de probabilidad.

Este despliegue demuestra cómo combinar **Python**, **FastAPI**, y técnicas de Machine Learning para crear una aplicación web funcional y accesible que puede detectar la presencia de patologías en voces humanas a partir de archivos de audio. La estructura modular del proyecto permite una fácil escalabilidad y mantenimiento, mientras que el uso de tecnologías web modernas garantiza una interfaz de usuario atractiva y efectiva.



**Detección de Voz Patológica**

**Sube tu archivo de audio**

Selecciona el modelo:

Modelo CNN

Carga un archivo de audio (.wav):

Choose File No file chosen

**Predecir**

© 2024 - Juan Acosta - Felipe Palacio

Figura 6.1: Página principal despliegue



Figura 6.2: Página de resultados despliegue

# Conclusiones

---

## 7.1. Conclusiones

En este proyecto, el objetivo fue desarrollar y experimentar con técnicas clásicas y de aprendizaje profundo dentro del ámbito del aprendizaje automático. Al final, se logró desarrollar un modelo que responde de manera destacable a la identificación de patologías en la voz. De esta forma, mediante una serie de experimentos, se comprobó y demostró la efectividad del aprendizaje automático en tareas complejas para los seres humanos.

Además, se evidenciaron las diferencias principales entre las técnicas clásicas y las de aprendizaje profundo, siendo las clásicas modelos de una sola capa, mientras que las técnicas de aprendizaje profundo emplean modelos multicapa que se ensamblan de acuerdo con el problema en particular. Otro aspecto notorio observado en los experimentos fue el cambio de enfoque en las técnicas de aprendizaje profundo, optando por representar los datos de manera distinta a un simple arreglo con la amplitud de la onda —como se había definido en las técnicas clásicas—, y pasando a una representación en forma de imagen. Esto se logró haciendo uso de una de las herramientas más poderosas de las matemáticas en el procesamiento de señales, con un fuerte trasfondo teórico y práctico: la transformada de Fourier.

La implementación de la aplicación permitió compartir los resultados finales de los experimentos realizados, proporcionando una herramienta práctica y asequible para el público con el fin de determinar algo tan complejo como una patología en la voz.

A futuro, los siguientes pasos del proyecto deben enfocarse en realizar una validación clínica de los resultados, que permita ajustar el modelo para su uso en contextos reales de manera más práctica y cómoda para pacientes y especialistas del sector salud. Desde luego, un estudio clínico es un paso crucial para que los resultados del experimento ganen credibilidad y aval en la comunidad médica y científica. En conclusión, este proyecto proporciona una comparativa entre técnicas clásicas y técnicas profundas de aprendizaje automático. Más allá de ello, permite vislumbrar el potencial general del aprendizaje automático en tareas que involucran la generación de datos, permitiendo identificar patrones en estructuras que para los seres humanos serían prácticamente indetectables y ahorrando tiempo, dinero y recursos en la ejecución de las mismas. Especialmente en el campo de la medicina, donde el tiempo para determinar un diagnóstico puede ser crucial en el tratamiento y recuperación de los pacientes, el aprendizaje automático tiene el potencial de reducir estas ventanas temporales, al igual que la brecha social que existe en el sistema de salud en países en vías de desarrollo o subdesarrollados.

## 7.2. Trabajo futuro

Como trabajo futuro, se plantea la integración de otras técnicas de aprendizaje automático emergentes en el campo de la investigación, como el aprendizaje automático cuántico y la exploración de modelos adicionales de aprendizaje profundo.

Además, es importante señalar que no se realizó ninguna validación clínica en este trabajo; por lo tanto, la validación clínica es un aspecto a considerar en investigaciones futuras, ya que su implementación podría permitir ajustar los hiperparámetros de los modelos. Asimismo, esta validación podría contribuir a obtener una mayor cantidad de datos para reentrenar los modelos y observar su comportamiento al ampliar el dataset.

### 7.2.1. Identificación del tipo de patología

La identificación multiclase de la patología específica padecida en cada muestra podría ser un objetivo en investigaciones futuras. Clasificar las muestras en función de una patología específica permitiría avanzar de la simple identificación de la presencia de patologías hacia la posibilidad de establecer un diagnóstico detallado de las mismas.

### 7.2.2. Uso de aprendizaje semisupervisado

El uso de aprendizaje semisupervisado en el preprocesamiento de los datos podría incrementar la fiabilidad de los resultados y mejorar las clasificaciones. El aprendizaje semisupervisado permitiría agrupar las muestras en clústeres, lo cual podría ayudar a establecer patrones más claros para los modelos de aprendizaje automático en el proceso de clasificación.

# Bibliografía

- [1] J. G. González, M. A. Álvarez, and A. Orozco, “Automatic assessment of voice quality in the context of multiple annotations,” pp. 6236–6238, 2015.
- [2] N. Q. Abdulmajeed, B. Al-Khateeb, and M. A. Mohammed, “A review on voice pathology: Taxonomy, diagnosis, medical procedures and detection techniques, open challenges, limitations, and recommendations for future directions,” *Journal of Intelligent Systems*, 2022.
- [3] V. Parsa and D. G. Jamieson, “Identification of pathological voices using glottal noise measures,” *Journal of Speech, Language, and Hearing Research*, vol. 43, pp. 469–485, 2000.
- [4] I. Cobeta, F. Núñez, and S. Fernández, *Patología de la voz*. SEORL PCF, 2013.
- [5] M. de los Ángeles Marín Abellán, “Perder la voz tras un cáncer de laringe,” *REVISTA DE ESTUDIOS FILOLÓGICOS*, no. 24, Enero 2013.
- [6] Johns Hopkins Medicine, “Voice disorders,” <https://www.hopkinsmedicine.org/health/conditions-and-diseases/voice-disorders>, [Online; accessed 6-11-2023].
- [7] Mayo Clinic Staff, “Voice disorders: Diagnosis and treatment,” <https://www.mayoclinic.org/diseases-conditions/voice-disorders/diagnosis-treatment/drc-20353024>, Oct. 2022, [Online; accessed 6-11-2023].
- [8] N. Roy, R. M. Merrill, S. Thibeault, R. A. Parsa, S. D. Gray, and E. M. Smith, “Prevalence of voice disorders in teachers and the general population,” *Journal of Speech, Language, and Hearing Research*, vol. 47, pp. 281–293, 4 2004.
- [9] S. Hegde, S. Shetty, S. Rai, and T. Dodderi, “A survey on machine learning approaches for automatic detection of voice disorders,” pp. 947.e11–947.e33, 11 2019.
- [10] M. Alhussein and G. Muhammad, “Voice pathology detection using deep learning on mobile healthcare framework,” *IEEE Access*, vol. 6, pp. 41 034–41 041, 7 2018.
- [11] “The 2012 acm computing classification system,” <https://dl.acm.org/ccs>, Association for Computing Machinery, 2012, accedido para la clasificación de temas en proyectos de investigación.
- [12] A. Shrestha and A. Mahmood, “Review of deep learning algorithms and architectures,” *IEEE Access*, vol. 7, pp. 53 040–53 065, 2019.
- [13] V. Nasteski, “An overview of the supervised machine learning methods,” *HORIZONS.B*, vol. 4, pp. 51–62, 12 2017.
- [14] D. Sun, J. Xu, H. Wen, and D. Wang, “Assessment of landslide susceptibility mapping based on bayesian hyperparameter optimization: A comparison between logistic regression and random forest,” *Engineering Geology*, vol. 281, 2 2021.

- [15] C. Kingsford and S. L. Salzberg, "What are decision trees?" 2008. [Online]. Available: <http://www.nature.com/naturebiotechnology>
- [16] L. Breiman, "Random forests," pp. 5–32, 2001.
- [17] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 5 2015.
- [18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [20] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*, vol. 4, 1992, pp. 950–957.
- [21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [23] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th international joint conference on Artificial intelligence*, vol. 2, 1995, pp. 1137–1143.
- [24] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing Management*, vol. 45, no. 4, pp. 427–437, 2009.
- [25] M. K. Gourisaria, R. Agrawal, M. Sahni, and P. K. Singh, "Comparative analysis of audio classification with mfcc and stft features using machine learning techniques," *Discover Internet of Things*, vol. 4, 12 2024.
- [26] K. Maharana, S. Mondal, and B. Nemade, "A review: Data pre-processing and data augmentation techniques," *Global Transitions Proceedings*, vol. 3, pp. 91–99, 6 2022.