Maestría en

Ciencia de Datos y Máquinas de Aprendizaje mención en Inteligencia Artificial

Trabajo de investigación previo a la obtención del título de

Magíster en Ciencia de Datos y Máquinas de Aprendizaje

con mención en Inteligencia Artificial

AUTORES:

Zambrano Chiliquinga Joselyn Isabel

Peralta Torres Juan Sebastián

Criollo Gallardo Sara Gimabel

Ochoa Guaraca Santiago David

TUTORES:

Iván Reyes

Alejandro Cortés López

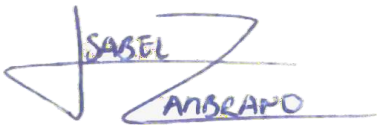
Comparación de Algoritmos de Clasificación para la Detección de Deepfakes en Señales de Audio

Quito, diciembre 2024

Certificación de autoría

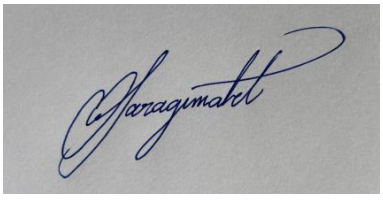
Nosotros, **Joselyn Isabel Zambrano Chiliquinga, Juan Sebastián Peralta Torres, Sara Gimabel Criollo Gallardo y Santiago David Ochoa Guaraca**, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

Cedemos nuestros derechos de propiedad intelectual a la Universidad Internacional del Ecuador (UIDE), para que sea publicado y divulgado en internet, según lo establecido en la Ley dePropiedad Intelectual, su reglamento y demás disposiciones legales.



Firma del graduando
Joselyn Isabel Zambrano Chiliquinga

Firma del graduando
Juan Sebastián Peralta Torres



Firma del graduando
Sara Gimabel Criollo Gallardo

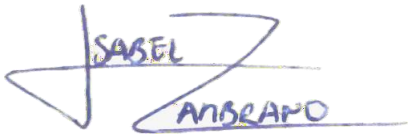
Firma del graduando
Santiago David Ochoa Guaraca

Autorización de Derechos de Propiedad Intelectual

Nosotros, **Joselyn Isabel Zambrano Chiliquinga, Juan Sebastián Peralta Torres, Sara Gimabel Criollo Gallardo y Santiago David Ochoa Guaraca**, en calidad de autores del trabajo de investigación titulado *Comparación de Algoritmos de Clasificación para la Detección de Deepfakes en Señales de Audio*,

autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, diciembre 2024



Firma del graduando
Joselyn Isabel Zambrano Chiliquinga

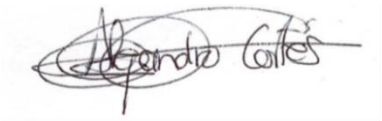
Firma del graduando
Juan Sebastián Peralta Torres

Firma del graduando
Sara Gimabel Criollo Gallardo

Firma del graduando
Santiago David Ochoa Guaraca

Aprobación de dirección y coordinación del programa

Nosotros, **Alejandro Cortés e Iván Reyes,** declaramos que los graduandos: **Joselyn Isabel Zambrano Chiliquinga, Juan Sebastián Peralta Torres, Sara Gimabel Criollo Gallardo y Santiago David Ochoa Guaraca** son los autores exclusivos de la presente investigación y que ésta es original, auténtica y personal de ellos.



Alejandro Cortés

Director de la Maestría en Ciencia de datos mención Inteligencia Artificial

Iván Reyes

Coordinador de la Maestría en Ciencia de datos mención en Inteligencia Artificial

Dedicatoria

A nuestras familias.

Agradecimientos

Estamos profundamente agradecidos con todas las personas que nos apoyaron durante este camino. A nuestros docentes de la maestría, gracias por todo el conocimiento impartido, ha sido su dedicación y empeño por lo que hacen una gran inspiración para nosotros.

A nuestros directores de tesis, gracias por su guía durante el desarrollo de este proyecto.

A los que compañeros que conformamos el equipo que realizó este proyecto, gracias por todo el esfuerzo y la perseverancia para realizar esta investigación.

Finalmente, gracias a nuestras respectivas familias y amigos por ser nuestro soporte, por brindarnos amor, tiempo y palabras de aliento, que nos motivaron, aún en los momentos más difíciles, a culminar con este reto

Resumen

El presente proyecto de investigación tiene como finalidad probar y comparar varios algoritmos de machine learning para la detección de deepfakes en señales de audio, esto con la finalidad de brindar una herramienta que clasifiqué con precisión audios reales y audios falsos.

En esta amplia investigación se proponen algoritmos de aprendizaje supervisado como Random Forest, Máquinas de Soporte Vectorial y Redes Neuronales Convolucionales, también se proponen algoritmos de aprendizaje semisupervisado como Self Training, Label Propogation y Label Spreading, y finalmente algoritmos de aprendizaje no supervisado de clusterización, autoencoders, asociación y detección de anomalías. Lo que permite determinar, para este caso de detección de deepfakes en audios, el modelo que mejor se comporta a nivel de predictibilidad, y que, en futuros estudios, podría convertirse en una herramienta que con robustez detecte deepfakes en audio en tiempo real. Se determinó en este proyecto que el modelo que mejor separa audios reales y falsos es una red neuronal convolucional de una dimensión, sin embargo, algoritmos de aprendizaje semi supervisado otorgaron buenos resultados para este problema, lo que los convierte en una técnica prometedora para la detección de deepfakes de sonido.

Palabras clave: Apendizaje Supervisado, Aprendizaje Semi-Supervisado, Aprendizaje no Supervisado, Random Forest, SVM, CNN, Self Training, Label Propagation, Label Spreading, Clustering, DBSCAN, Autoencoders.

Abstract

The purpose of this research project is to test and compare several machine learning algorithms for detecting deepfakes in audio signals, with the aim of providing a tool that accurately classifies real audio and fake audio.

In this extensive research, supervised learning algorithms such as Random Forest, Support Vector Machines and Convolutional Neural Networks are proposed, semi-supervised learning algorithms such as Self Training, Label Propogation and Label Spreading are also proposed, and finally unsupervised learning algorithms for clustering. autoencoders, association and anomaly detection. This allows us to determine, for this case of detecting deepfakes in audio, the model that performs best in terms of predictability, and which, in future studies, could become a tool that robustly detects deepfakes in audio in real time. It was determined in this project that the model that best separates real and fake audio is a one-dimensional convolutional neural network; however, semi-supervised learning algorithms provided good results for this problem, which makes them a promising technique for detection. of sound deepfakes.

Keywords: Supervised learning, Semi-Supervised learning, Unsupervised Learning, Random Forest, SVM, CNN, Self Training, Label Propagation, Label Spreading, Clustering, DBSCAN, Autoencoders.

Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El rápido avance en la IA, aprendizaje automático y aprendizaje profundo han contribuido de manera positiva en varias áreas de la salud, industria, educación, etc. De igual manera la IA generativa ha llegado a revolucionar la forma en la que estudiamos y trabajamos, ayudando a agilizar tareas repetitivas, proporcionando ideas originales, generando código de programación, y también ha revolucionado la forma en la que se generan y manipulan contenidos digitales, pues ha dado lugar a nuevas técnicas y herramientas para crear audios, imágenes y vídeos. Aunque la tecnología se ha utilizado principalmente en aplicaciones legítimas como el entretenimiento y la educación, los usuarios malintencionados también la han explotado para fines ilegales, (Rana, 2022).

Los deepfakes utilizan técnicas avanzadas de aprendizaje automático y aprendizaje profundo (DL) para crear vídeos, imágenes y audios que pueden ser extremadamente realista, esta tecnología ayuda, por ejemplo, a crear contenido audiovisual de una manera fácil, rápida y menos costosa para comunicaciones de marketing para empresas, sin embargo, esta tecnología también se ha usado para hacer uso ilegítimo e indebido de la imagen personal y puede incluso representar una amenaza para la democracia, la seguridad nacional y la privacidad. Esto genera diversos problemas, desde engañar a la opinión pública hasta utilizar pruebas manipuladas en un tribunal, (Heidari, 2024).

La utilización de herramientas generadas para clonar la voz humana se encuentra en auge, actualmente, lo que ha dado lugar a una nueva tecnología dentro del Deepfake conocida como Audio Deepfakes (ADs), que son herramientas sintetizadas por IA con la capacidad de generar voces convincentes. Aunque fueron introducidas para mejorar la vida humana, como en audiolibros, los ADs se han utilizado también para poner en riesgo la seguridad pública. Por ello, los ADs han llamado recientemente la atención de los investigadores, y se han desarrollado métodos basados en Aprendizaje Automático (ML) y Aprendizaje Profundo (DL) para detectarlos, (Almutairi, 2022)

Recientemente se han desarrollado herramientas sintetizadas por IA con la capacidad

de generar voces convincentes, (Lyu, 2020). Sin embargo, aunque estas herramientas fueron introducidas para ayudar a las personas, también se han utilizado para difundir desinformación a nivel mundial a través de audio, (Johnson, 2021),y su uso malicioso ha generado temor en torno al ”Audio Deepfake”. Diariamente se transmiten cantidades masivas de grabaciones de voz en Internet, y detectar su falsedad es una tarea difícil. Sin embargo, los atacantes de AD han dirigido sus ataques no solo a individuos y organizaciones, sino también a políticos y gobiernos, (Suwajanakorn, 2017).

Por lo tanto, la investigación sobre métodos eficientes para la detección de deepfakes se vuelve imperativa. A medida que avanzan las técnicas de generación de estos contenidos, los enfoques tradicionales de detección son insuficientes para afrontar las nuevas y más sofisticadas formas de manipulación. En consecuencia, este proyecto de titulación aborda dicha problemática, explorando y proponiendo soluciones basadas en modelos de inteligencia artificial que permitan la identificación precisa de deepfakes, contribuyendo así a mitigar los riesgos asociados a su uso.

Problema de Investigación

La Inteligencia Artificial (IA) tiene una proyección inmensurable de cara al futuro, con implementaciones únicas que pueden cambiar la cotidianidad del ser humano. Sin embargo, esta tecnología bajo el mando de entidades que busquen perjudicar y/o extorsionar resulta ser más un arma que una herramienta de ayuda al ser humano. En consecuencia, se tiene evidencia de todo tipo en donde se falsifica toda clase de contenido digital tales como imágenes, audios, vídeos, etc., mediante IA.

La generación de audios falsos que imitan voces reales con gran precisión puede tener consecuencias perjudiciales significativas, desde la desinformación hasta el daño a la reputación de individuos y organizaciones, es por eso que la implementación de técnicas avanzadas para la detección de deepfakes de audio es crucial para salvaguardar la integridad de la información y mantener la confianza pública.

Además, la detección efectiva de estos contenidos sintéticos previene su uso malintencionado, tales como el chantaje, la suplantación de identidad y la difamación,

protegiendo así la privacidad y los derechos individuales. Este enfoque proactivo en la identificación y mitigación de deepfakes contribuye a una mayor seguridad en el entorno digital, ya que la generación y propagacion de audios falsos afecta la veracidad de la información que se transmite a las personas, y provoca inseguridad, ya que en la actualidad las noticias digitales, imágenes, vídeos, entre otros, son los medios en donde la mayoría de la población está acostumbrada a captar información y el deepfake puede generar realidades falsas a partir de modificaciones de la realidad, (Capistrán, 2020)

Adicionalmente, el desarrollo de tecnologías avanzadas para la detección de deepfakes de audio no solo promueve el avance en el campo de la inteligencia artificial y el aprendizaje automático, sino que también ofrece nuevas perspectivas y metodologías para abordar problemas relacionados con la autenticidad del contenido. Este proyecto también tiene un impacto social y educativo significativo, al proporcionar herramientas y conocimientos para combatir las amenazas digitales emergentes.

En este trabajo de titulación se implementarán varias técnicas de apredizaje automático y aprendizaje profundo para la detección de deepfakes de audio en busca del algoritmo más preciso para la detección de audios generados con IA, esto buscando mejorar la seguridad en las plataformas digitales y preservar la veracidad de la información.

Objetivos

Objetivo General

Desarrollar y comparar e identificar los modelos de Machine Learning, con mayor precisión al momento de detectar deep fakes de señales de audio, contribuyendo la mejora en la seguridad en plataformas digitales y preservar la veracidad de la información.

Objetivo Específicos

Aplicar técnicas de Aprendizaje Supervisado utilizando algoritmos de clasificación como SVM, Random Forest y Redes Neuronales para entrenar modelos capaces de detectar deep fakes en audios con etiquetas predefinidas, logrando una precisión mínima del 80% en un plazo de 3 meses.

Aplicar técnicas de Aprendizaje No Supervisado utilizando algoritmos como K-

means, DBSCAN y Autoencoders para entrenar modelos que identifique deep fakes en audios, buscando alcanzar un valor mínimo de 0.5 en la métrica de Silhouette Score en un plazo de 3 meses.

Aplicar técnicas de Aprendizaje Semi-Supervisado como Self-Learning, Label-Spreading y Label-Propagation, integrando un conjunto limitado de datos etiquetados y un mayor volumen de datos no etiquetados, con el objetivo de detectar deep fakes en audios en un plazo de 3 meses. La efectividad se medirá con la métrica de F1 Score, buscando alcanzar un valor mínimo de 0.75.

Comparar el rendimiento de métodos de aprendizaje supervisado, no supervisado y semisupervisado en la detección de deepfakes en audio, evaluando métricas clave como precisión, sensibilidad finalizando el análisis comparativo en un plazo de 3 meses.

Capítulo 2

En esta sección se presentan definiciones y conceptos teóricos necesarios para comprender la metodología utilizada en la construcción y comparación de los modelos de detección de deepfakes en audios, la cual consta de las siguientes etapas:

1. Análisis y limpieza de las variables obtenidas de los audios.
2. Se extraen las características espectrales de los audios usando la libreía librosa de Python.
3. Entrenamiento de los modelos de machine learning y deep learning.
4. Se entrena un modelo de random forest que clasifiquen los audios en FAKE/REAL.
5. Se ajusta un modelo de máquina de soporte vectorial para dos clases FAKE/REAL.
6. Se realizó una simulación en la que existen audios sin clasificar como FAKE o REAL, y a partir de este dataset simulado se entrenaron tres modelos de aprendizaje semi-supervisado, estos son: Self-Learning, Label Propagation y Label Spreading.
7. Se crea la arquitectura y el ajuste de una red neuronal convolucional para clasificar los audios en FAKE/REAL.
8. Finalmente, se compararán los tres modelos obtenidos y se optará por aquel que produzca mejores resultados en términos de poder predictivo.

Para el entendimiento de cada una de las etapas a continuación se detallan las nociones y base teórica de cada una de las técnicas mencionadas.

Random Forest

El algoritmo basado en Random Forest (RF) es capaz de realizar tareas de cuantificación y clasificación. Desarrollado por (Breiman, 2021), consiste en una colección de clasificadores tipo árbol estructurados de tal forma que $h(x, \theta_k), k = 1,$, donde θ_k es un vector independiente aleatorio y cada árbol emite un único voto para la clase más popular dado una entrada x.

Se utiliza randomización para crear un número largo de árboles de decisión, obteniendo una única salida usando votación para problemas de clasificación y valores promedio para problemas de regresión, (Rigatti, 2017).

Según (Yang, 2016), el algoritmo de esta metodología es aplicado con los siguientes pasos:

1. Para $b = 1$ hacia B:
    a) Extraiga una muestra Z^*de tamaño N del conjunto de datos de entrenamiento.
    b) Desarrolle un árbol de bosque aleatorio T_ba partir de los datos, mediante la repetición recursiva de los siguientes pasos para cada nodo terminal del árbol, hasta que el tamaño mínimo del nodo n_{min} se consiga.
    c) Seleccione m variables al azar de las variables p.
    d) Escoja la mejor variable o punto de división a lo largo de m.
    e) Divida el nodo padre en dos nodos hijos.
2. Desarrolle el ensamble de árboles.

Por lo tanto, para desarrollar una predicción de acuerdo con un nuevo punto x, se tiene las siguientes ecuaciones:

Regresión:

$$\hat{f}\mathrm{rf}^{B}(x) = \frac{1}{B}\sum b = 1^{B} T_b(x)$$

Clasificación

Para una predicción de clase $\widehat{C_b}(x)$ del b-ésimo árbol del bosque aleatorio, se tiene que:

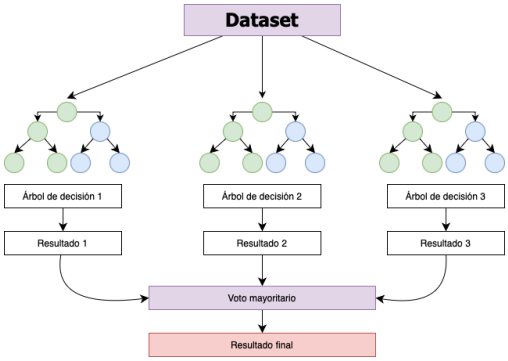
$$\widehat{C_{\mathrm{rf}}^{B}}(x) = \text{ majority vote } \left\{\widehat{C_b}(x)\right\}_1^B$$

Entre las principales ventajas de este algoritmo se encuentran que tiene una capacidad de trabajo eficiente con set de datos muy grandes, una precisión alta al momento de realizar las predicciones, cada árbol puede ser entrenado en paralelo al ser independiente uno del otro. Como desventaja, la más importante puede ser la interpretabilidad de dicho algoritmo, ya que cada salida puede tener diferentes posibles caminos y, en modelos

extensos, puede comportarse como una caja negra, es decir, obtener respuestas que no se saben cómo se consiguieron, (Rodríguez Copado, 2024).

Figura 1

Funcionamiento de los modelos



Nota: Se observa en la gráfica el funcionamiento de los modelos Random Forest.

Máquinas de Soporte Vectoriales (SVM)

Este conjunto de algoritmos pertenece a la familia de clasificadores lineales generalizados y utlizan métodos de Machine Learning (ML) con aprendizaje supervisado para clasificación y regresión. Además, son capaces de maximizar la exactitud de predicción mientras que automáticamente se evita el sobreajuste del modelo, (Jakkula, 2066).

Los SVMs pueden ser definidos como sistemas que utilizan separadores lineales, conocidos como hiperplanos, dentro del espacio original de entradas o en un espacio transformado. Esto dependiendo si las entradas son linealmente separables o no. Dependiendo de la complejidad de separación en los espacios transformados, que son normalmente de muy alta dimensión, la búsqueda del hiperplano se realizará mediante las funciones kernel, (Suarez, 2014).

Según (Suarez, 2014), las funciones kernel K asignan a cada par de elementos de un espacio de entrada X, un valor real correspondiente al producto escalar de las proyecciones de dichos elementos en un nuevo espacio de características:

$$K: X \times X \rightarrow R$$

Dentro de las funciones kernel, se tiene:

Kernel lineal:

$$K_L(\boldsymbol{x}, \boldsymbol{x}') =< \boldsymbol{x}, \boldsymbol{x}' >$$

Kernel polinómico:

$$K_P(\boldsymbol{x}, \boldsymbol{x}') = [\gamma < \boldsymbol{x}, \boldsymbol{x}' > +\tau]^p, \gamma > 0$$

Kernel Gaussiano:

$$K_G(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma|\boldsymbol{x} - \boldsymbol{x}'|^2) \equiv \exp(-\gamma < (\boldsymbol{x} - \boldsymbol{x}'), (\boldsymbol{x} - \boldsymbol{x}') >), \gamma > 0$$

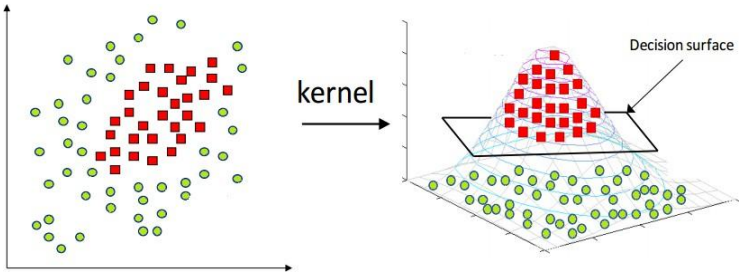
Kernel Sigmoidal:

$$K_S(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\gamma < \boldsymbol{x}, \boldsymbol{x}' > +\tau)$$

Donde $\gamma\$, \$\tau$ y p son parámetros de kernel.

Figura 2

Funcionamiento del Kernel en los SVM.



Nota: Se observa en la gráfica el funcionamiento de los modelos Random Forest. Fuente: (NewTechDojo, 2017)

Aprendizaje Semisupervisado

El aprendizaje semisupervisado, una técnica híbrida que combina elementos del aprendizaje supervisado y no supervisado ha surgido como una solución atractiva para abordar el desafío de la escasez de datos etiquetados en muchos problemas de machine learning (Chapelle, 2006). A diferencia del aprendizaje supervisado, que requiere grandes cantidades de datos etiquetados para entrenar modelos precisos, el aprendizaje semisupervisado aprovecha tanto datos etiquetados como no etiquetados. Esta combinación permite a los algoritmos construir modelos más robustos y generalizables, especialmente en escenarios donde la adquisición de datos etiquetados resulta costosa o laboriosa (Zhu, 2022). La idea central del aprendizaje semisupervisado es que los datos no etiquetados pueden

proporcionar información valiosa sobre la estructura subyacente de los datos, complementando así la información proporcionada por los datos etiquetados.

El aprendizaje semisupervisado ha encontrado aplicaciones en una amplia gama de campos, incluyendo la clasificación de imágenes (por ejemplo, en diagnóstico médico), el procesamiento del lenguaje natural (como en análisis de sentimientos), las recomendaciones personalizadas, etc. En estas áreas, esta técnica ha demostrado su eficacia al mejorar la precisión de los modelos y reducir la dependencia de grandes conjuntos de datos etiquetados.

Self - Training

El método de Self-Training es una estrategia iterativa dentro del aprendizaje semisupervisado que aprovecha un modelo inicial entrenado con datos etiquetados para generar pseudoetiquetas para los datos no etiquetados (Zhu, 2022).

Este proceso iterativo consiste en entrenar repetidamente el modelo con los datos etiquetados originales y las nuevas pseudoetiquetas hasta alcanzar un criterio de convergencia. Esta técnica, aunque sencilla, ha demostrado ser eficaz en diversas aplicaciones, (Chapelle, Semi-supervised learning., 2006).

Formalmente, dado un conjunto de datos etiquetados:

$$D_l = (x_i, y_i)_{i=1}^{n_l}$$

y un conjunto de datos no etiquetados:

$$D_u = x_{i_{n_l+1}}^{n}$$

el algoritmo de Self-Training puede describirse como lo siguente:

1. **Inicialización:** Entrenar un modelo inicial f utilizando únicamente D_l.
2. Iteración:
    a) **Generación de pseudoetiquetas:** Asignar pseudoetiquetas a los datos no etiquetados:

$$y^i = f(x_i)\ para\ todo\ x_i \in D_u.$$

    b) **Reentrenamiento**: Reentrenar el modelo utilizando tanto los datos

etiquetados como las pseudoetiquetas:

$$f \leftarrow \arg\min_{f} \sum_{i=1}^{n_l} L\big(y_i, f(x_i)\big) + \lambda \sum_{i=n_l+1}^{n} L\big(\hat{y}_i, f(x_i)\big)$$

donde L es una función de pérdida y λ es un hiperparámetro que equilibra la importancia de los datos etiquetados y no etiquetados.

Ventajas y Desafíos:

Simplicidad: Su implementación es relativamente sencilla, lo que lo convierte en un punto de partida accesible para muchos investigadores.

Escalabilidad: Es aplicable a grandes conjuntos de datos, lo cual es una ventaja en muchas aplicaciones reales.

Sensibilidad a errores: Los errores en las primeras iteraciones pueden propagarse y afectar el rendimiento final del modelo, como lo han señalado diversos estudios, (Lee, 2003).

Suposición de pureza: Asume que los datos no etiquetados pertenecen a las mismas clases que los datos etiquetados, lo cual puede no ser siempre cierto en la práctica.

Label propagation

Label Propagation es una técnica de aprendizaje semisupervisado que se basa en la premisa de que datos similares tienden a tener etiquetas similares. Esta técnica construye un grafo donde los nodos representan los datos y los bordes cuantifican la similitud entre ellos. Al propagar las etiquetas conocidas de los datos etiquetados a través de este grafo, podemos inferir las etiquetas de los datos no etiquetados. Como señalan (Zhou, 2003), esta técnica aprovecha la estructura inherente en los datos para mejorar la precisión de la clasificación.

Matemáticamente, el proceso de propagación se modela como una actualización iterativa de la matriz de etiquetas:

$$Y^{(t+1)} = D^{-1} W Y^{(t)}$$

Donde Y (t) representa la matriz de etiquetas en la iteración t, D es la matriz diagonal de grados, y W es la matriz de pesos del grafo. Esta formulación, que captura la idea de

difusión de etiquetas a través de los vecinos en el grafo, es fundamental para el algoritmo de Label Propagation.

La eficacia de Label Propagation depende en gran medida de la calidad del grafo construido. Un grafo bien diseñado captura las relaciones de similitud entre los datos de manera precisa, permitiendo una propagación de etiquetas más efectiva. Sin embargo, como señalan (Zhu, 2022), la construcción de un grafo óptimo puede ser un desafío, especialmente para grandes conjuntos de datos.

Label Spreading

Label Spreading es una extensión del algoritmo de Label Propagation que introduce un grado adicional de flexibilidad al proceso de propagación de etiquetas. A diferencia de Label Propagation, Label Spreading incorpora un parámetro de suavizado que controla la influencia de las etiquetas iniciales y la fuerza con la que las etiquetas se difunden a través del grafo (Zhou, 2003).

Matemáticamente, la actualización de las etiquetas en Label Spreading se expresa como:

$$Y^{(t+1)} = (1 - \alpha)D^{-1}WY^{(t)} + \alpha Y^{(0)}$$

Donde:

- Y (t) es la matriz de etiquetas en la iteración t.
- D es la matriz diagonal de grados.
- W es la matriz de pesos del grafo.
- α es el parámetro de suavizado, un valor entre 0 y 1.
- Y (0) es la matriz de etiquetas inicial.

El parámetro de suavizado α juega un papel fundamental en el comportamiento del algoritmo. Un valor de α cercano a 1 indica que se da más importancia a las etiquetas iniciales, lo que resulta en una propagación de etiquetas más suave y menos influenciada por la estructura del grafo. Por otro lado, un valor de cercano a 0 enfatiza la influencia de los vecinos en la actualización de las etiquetas, lo que puede llevar a una propagación más local y a una mayor sensibilidad a la estructura del grafo.

Redes Neuronales Artificiales (ANNs)

Las redes neuronales están inspiradas en la sofisticada funcionalidad del cerebro humano, donde cientos de miles de millones de neuronas interconectadas procesan información en paralelo.

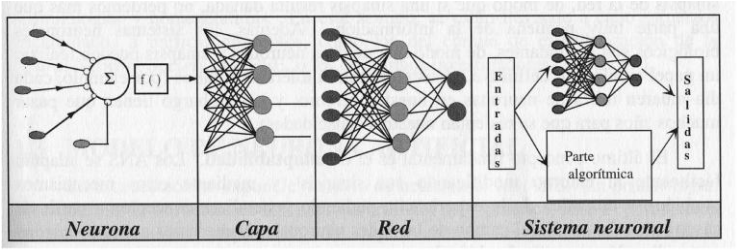
Según (Larranaga, 1997), hay tres conceptos claves que se deben emular en un sistema neuronal artificial:

- **Procesamiento paralelo:** En todos los procesos cognitivos los miles de millones de neuronas que intervienen están operando al mismo tiempo, es decir, en paralelo.
- **Memoria distribuida:** Mientras que en computadores la información está en posiciones de memoria bien definidas, en las redes neuronales biológicas dicha información está distribuida por la sinapsis de la red, existiendo una redundancia en el almacenamiento, para evitar la pérdida de información en caso de que una sinapsis resulte dañada.
- **Adaptabilidad al entorno:** Está adaptabilidad al entorno permite aprender de la experiencia y hace posible generalizar conceptos a partir de casos particulares.

Según (Larranaga, 1997), el elemento básico en todo este sistema neuronal biológico es la neurona. Un sistema neuronal biológico está compuesto por millones de neuronas organizadas en capas. En la imitación de este sistema neuronal biológico, por medio de un sistema neuronal artificial, el elemento principal será la neurona artificial, la cual se organizará en capas y varias capas constituirán una red neuronal. Es así como una red neuronal, junto a las interfaces de entrada y salida, conforman el sistema global del proceso de aprendizaje.

Figura 3

Sistema global del funcionamiento de una red neuronal



Nota: Se describe un sistema global del funcionamiento de una red neuronal. Fuente: (Larranaga, 1997).

Como lo podemos ver en la figura 3, la arquitectura típica de una red neuronal artificial consiste en una capa de entrada de neuronas, una o más capas ocultas de neuronas, y una capa final de neuronas de salida. Cada conexión entre las capas está asociada a pesos. La salida, h_i, de la neurona i, en la capa oculta es:

$$h_i = \sigma\left(\sum_{j=1}^{N} V_{ij}\, x_j + T_i^{hid}\right)$$

donde σ(.) es la función de activación (o función de transferencia), N es el número de neuronas de entrada, V_{ij}los pesos, x_j las neuronas de entrada, y T_i^{hid} el umbral en término de las neuronas ocultas. El propósito de la función de activación es, además de introducir no linealidad en la red neuronal, limitar el valor de la neurona para que la red neuronal no se paralice debido a valores divergentes. En este trabajo usaremos, como función de activación, la función sigmoide (o función logística) dado que es un problema de clasificación.

Redes Neuronales Artificiales Convolucionales (CNNs)

Según (Thomas, 2014), las Redes Neuronales Convolucionales son similares a las Redes Neuronales Artificiales en el sentido de que están compuestas por neuronas que automáticamente se optimizan mediante el aprendizaje, siendo la principal diferencia la capa adicional de extracción de características.

Las CNN son un tipo de arquitectura de red neuronal diseñada especialmente para procesar datos con estructuras de cuadrícula, como las imágenes. Estas redes han sido ampliamente utilizadas en tareas de visión por computadora debido a su capacidad para capturar patrones espaciales y jerárquicos en los datos, (Yamashita, 2018). Aunque

tradicionalmente se han usado en el procesamiento de imágenes, también pueden aplicarse en datos de audio. En el caso del audio, donde las señales tienen una dimensión temporal, se emplean capas convolucionales unidimensionales (Conv1D). Estas capas permiten extraer características de la señal en el dominio temporal, lo cual es útil para tareas como la detección de deepfakes en audios. A continuación, se explican los componentes clave de una CNN para detección de deepfakes en audios.

Capas de Convolución 1D (Conv1D)

Dado que se aplica una CNN a señales de audio, las capas de convolución 1D son de utilidad para la extracción de características. En lugar de aplicar un filtro en dos dimensiones (como se haría en imágenes), en Conv1D el filtro se desplaza solo en una dimensión, a lo largo del tiempo. La operación de convolución para una señal unidimensional *x* y un kernel *K* se define como:

$$S(t) = \sum_i x(t+i) \cdot K(i)$$

donde, *S*(*t*) es el valor de salida en la posición $t, x(t+i)$ es la señal de entrada desplazada por *i* y *K*(*i*) representan los valores del Kernel que se ajustan mediante el entrenamiento.

Esto permite que el modelo aprenda patrones locales de las señales de audio, como cambios de amplitud y frecuencia que podrían indicar anomalías relacionadas con deepfakes.

Función de activación

Luego de la convolución 1D, se aplica una función de activación no lineal, ReLU (Rectified Linear Unit), para introducir no linealidad en la red y permitir que aprenda relaciones más complejas, (He, 2018) (Naeem, 2023). La función ReLU se define como:

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

Esto convierte a cero los valores negativos, dejando los valores positivos sin cambios.

ReLU ayuda a que la CNN aprenda patrones complejos en la señal de audio y hace

que el entrenamiento sea más eficiente al reducir problemas como el desvanecimiento de gradientes.

Capas de Pooling 1D

Las capas de pooling 1D (o agrupamiento) se utilizan para reducir la dimensión temporal de la señal, manteniendo solo la información relevante. El tipo más común es el max pooling, donde se selecciona el valor máximo dentro de una región de la señal. En este caso de una señal unidimensional, el max pooling se define como:

$$P(t) = \backslash operatorname * max_i S(t+i)$$

donde, $P(t)$ es el valor de salida del pooling en la posición t y $S(t+i)$ representa la región de entrada sobre la cual se aplica el max pooling.

Este método ayuda a que la CNN sea invariante a pequeñas variaciones en el tiempo, es decir, que no sea afectada por cambios leves en la posición temporal de los patrones en la señal de audio.

Capas Completamente Conectadas

Luego de varias capas de convolución y pooling, la salida se ”aplana” y se pasa a una o más capas completamente conectadas. En este proceso, cada neurona se conecta a todas las neuronas de la capa anterior, permitiendo una combinación de las características aprendidas.

Matemáticamente, la salida de una neurona en una capa completamente conectada se expresa como:

$$z = \sum_i w_i \, x_i + b$$

donde, z es la salida de la neurona, w_i son los pesos ajustados durante el entrenamiento, x_i son las entradas de la neurona y b es el sesgo, también aprendido durante el entrenamiento.

Estas capas completamente conectadas permiten combinar todas las características extraídas para realizar la clasificación o detección de deepfakes en el audio.

Optimización

El entrenamiento de una CNN para la detección de deepfakes en audio se concentra en minimizar una función de pérdida que cuantifica la diferencia entre las predicciones del modelo y las etiquetas verdaderas. Para tareas de clasificación binaria (”FAKE” vs. ”REAL”), es común utilizar la entropía cruzada binaria.

La optimización se realiza mediante retropropagación y algoritmos como el descenso de gradiente. A través de la retropropagación, el modelo ajusta los pesos (incluidos los valores del kernel en las capas de convolución) para reducir la diferencia entre las predicciones y las etiquetas verdaderas. Matemáticamente, la actualización de pesos se expresa como:

$$w = w - \eta \frac{\partial L}{\partial w}$$

donde, w es el peso actual, η es la tasa de aprendizaje y $\frac{\partial L}{\partial w}$ es el gradiente de la función de pérdida L con respecto a w.

Aprendizaje no supervisado

Cuando los científicos de datos utilizan conjuntos de datos para entrenar algoritmos, comienza el proceso de aprendizaje no supervisado. Estos conjuntos de datos no incluyen puntos de datos etiquetados o clasificados. El propósito del aprendizaje del algoritmo es encontrar patrones en el conjunto de datos y clasificar los puntos de datos según esos patrones. Los problemas de agrupamiento, asociación, detección de anomalías y autoencoders son cuatro tipos de desafíos del aprendizaje no supervisado (Naeem, 2023).

El aprendizaje no supervisado abarca un conjunto de herramientas estadísticas diseñadas para contextos en los que solo se dispone de un conjunto de características $X_1, X_2, \dots, X_n$ medidas en n observaciones. No se centra en la predicción, ya que no se cuenta con una variable respuesta asociada Y, (James, 2023).

El aprendizaje no supervisado se realiza con frecuencia como parte de un análisis exploratorio de datos. Además, puede ser difícil evaluar los resultados obtenidos de los métodos de aprendizaje no supervisado, ya que no existe un mecanismo universalmente aceptado para realizar validación cruzada o para validar resultados en un conjunto de datos

independiente, (Choudhury, 2022).

Tipos de Aprendizaje No Supervisado

1. **Clustering o Análisis de Agrupamiento**

   Para (Rolf, 2024) el **clustering** o análisis de agrupamiento es la práctica de clasificar elementos en grupos. Este proceso puede dividirse en varias formas, incluidas las siguientes:

   a) **Particionamiento:** Los datos se dividen de manera que cada elemento de información solo pueda pertenecer a un único grupo (cluster). Esto también se conoce como agrupamiento exclusivo. Un ejemplo típico de este método es el algoritmo **K-Means**.

   b) **Agrupamiento Jerárquico:** Cada punto de datos inicia como un grupo individual, y mediante conexiones iterativas entre los dos clusters más cercanos, se reduce el número de grupos. Este método organiza los datos en conjuntos jerárquicos y puede ser **aglomerativo** o **divisivo**.

   c) **Agrupamiento Superpuesto:** Permite que cada punto de datos pertenezca a dos o más categorías con distintos grados de afinidad como en el agrupamiento con **K-Means**.

   d) **Agrupamiento Probabilístico:** Genera los grupos utilizando distribuciones de probabilidad.

2. **Asociación**

El enfoque de aprendizaje no supervisado de Aprendizaje de Reglas de Asociación (ARL) se utiliza para descubrir asociaciones entre variables en grandes conjuntos de datos. A diferencia de algunos métodos de aprendizaje automático, ARL puede aceptar puntos de datos no numéricos. En resumen, ARL se centra en cómo están vinculadas ciertas variables. Por ejemplo, las personas que compran una motocicleta tienen más probabilidades de comprar un casco. Es posible ganar dinero formando este tipo de conexiones. Supongamos que los consumidores que compran el producto X también compran el producto Y, y una tienda en línea puede sugerir el producto Y a cualquiera que compre el producto X.

Internamente, se utilizan declaraciones para aprender las leyes de asociación. Estas declaraciones pueden resaltar conexiones entre conjuntos de datos dispares. Se utilizan medidas como **soporte** y **confianza** cuando se identifican patrones o relaciones. El soporte determina la frecuencia con la que aparece la relación si / entonces en la base de datos. El número de veces que la relación si/entonces ha sido determinada como legítima se llama **confianza**. La regla de asociación permite el análisis de carritos de compras y minería de uso en línea, (Ige, 2022).

3. Detección de Anomalías

Cualquier procedimiento que descubra puntos atípicos (outliers) en un conjunto de datos se conoce como **detección de anomalías**. Estas anomalías pueden sugerir actividad inusual en una red, un sensor defectuoso o datos que necesitan ser limpiados antes del análisis. Cuando los modelos de datos se desvían o se alejan de los modelos usuales, se considera una anomalía. Un patrón inusual de tráfico de red, por ejemplo, indica que un sistema comprometido está transfiriendo datos sensibles a un servidor no autorizado. Las anomalías se identifican o predicen encontrando o anticipando puntos de datos que difieren del modelo estándar. Algunas aplicaciones de la detección de anomalías incluyen la detección de intrusiones, seguros, detección de fraudes y vigilancia militar, (Pinto, 2023).

4. Autoencoders

Los **autoencoders** son un enfoque de aprendizaje no supervisado que utiliza redes neuronales para realizar aprendizaje de representaciones. Crearemos una arquitectura de red neuronal con un cuello de botella que obliga a la red a usar una representación comprimida del conocimiento del input original. Esta compresión y posterior reconstrucción sería complicada si las propiedades del input no estuvieran relacionadas. Si los datos tienen alguna estructura (por ejemplo, correlaciones entre los atributos de entrada), esta estructura puede ser aprendida y utilizada al pasar los datos a través del cuello de botella de la red. La presencia de un cuello de botella de información es una característica fundamental del diseño de nuestra red; sin él, nuestra red podría aprender rápidamente a almacenar los datos de entrada simplemente pasándolos a través de la red, (Jafari, 2020).

Estadísticos de evaluación de modelos de clasificación

Matriz de confusión

La matriz de confusión es un método que permite la visualización de los resultados de un algoritmo de clasificación, donde la variable objetivo puede tener dos o más clases. Las columnas de la matriz representan el número de predicciones de cada clase, y las filas representan los valores observados de cada clase. Para un algoritmo de clasificación binario, se tiene una tabla con cuatro combinaciones de valores predichos y reales, como se presenta en la figura 4.

Los elementos de esta matriz son:

- **TP:** Verdaderos positivos, es decir, el número de veces que el valor real y predicho es positivo.
- **TN:** Verdaderos negativos, es decir, el número de veces que tanto el valor real como el predicho es negativo**.**
- **FP:** Falsos positivos, el número de veces que erróneamente la prueba predijo un valor positivo cuando el valor real de la clase es negativo.
- **FN:** Falsos negativos, el número de veces que erróneamente la prueba predijo un valor negativo cuando el valor real de la clase es positivo.

Figura 4

Matriz de confusión

		Clase predicha	
		Positiva	**Negativa**
Clase real	**Positiva**	TP	FN
	Negativa	FP	TN

Nota: Este gráfico describe el funcionamiento de una matriz de confusión.

A partir de la matriz de confusión es posible calcular otras métricas de evaluación de modelos de clasificación, los que se usan en este proyecto son:

Exactitud: Es la proporción de valores correctamente predichos.

$$Exactitud = \frac{TP + TN}{TP + FP + TN + FN}$$

Precisión: Es la proporción de valores valores predichos positivos que son correctos.

$$Precisión = \frac{TP}{TP + FP}$$

Sensibilidad: Es la proporción de verdaderos positivos, muestra la capacidad del modelo para predecir los valores positivos.

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

F1 Score: La media armónica de precisión y sensibilidad, resumen la precisión y sensibilidad para determinar la precisión de un modelo. Esta métrica es muy útil para casos donde se tienen clases desbalanceadas.

$$F1 - Score = \frac{2 * Precisión * Sensibilidad}{Precisión + Sensibilidad}$$

Desarrollo

Descripción de los datos

Para este estudio se consideran las voces de 8 personas, en la tabla 1 se puede conocer el origen de los audios originales. Estos han pasado por un proceso de conversión de voz, que transformó la voz de una persona en la voz de otra, sin cambiar el mensaje del audio. La base de datos se obtuvo del trabajo realizado por (Bird, 2023)

A partir de estos se obtuvieron 56 audios deepfakes, es decir, de cada audio se generó la conversión de voz a la de las otras siete personas.

Estos audios son representativos de la realidad, es decir, dado que algunos son discursos dados en público, o entrevistas, tienen ruido de fondo tales como aplausos o gritos de apoyo.

Tabla 1

Descripción de los audios originales

Individuo	**Tiempo (mm:ss)**
Joe Biden	10:00
Linus Sebastian	9:30
Margot Robbie	1:19
Elon Musk	10:00
Barack Obama	10:00
Ryan Gosling	1:33
Taylor Swift	10:00
Donald Trump	10:00
Total	**62:22**

Nota: Este tabla describe los tiempos que duran los audios originales.

Extracción de datos

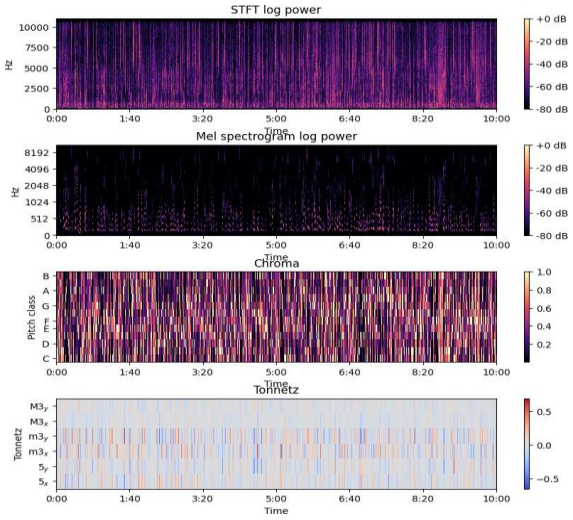
Considerando que se tienen 64 audios entre reales y falsos se extraen las características más importantes de cada audio, estas permiten clasificar los audios basados en sus características tonales, frecuenciales y temporales.

Para la extracción de variables se usó la librería *librosa* de Python, que permite extraer representaciones espectrales de los audios, siendo la mayoría de estas características basadas en la transformada de Fourier de corto plazo, (McFee, 2015).

En la figura 5, el primer gráfico es la transformada de Fourier de corto plazo, el segundo es el espectrograma de Mel correspondiente, tercero es el cromagrama correspondiente y cuarto son las características de Tonnetz

Figura 5

Descomposición espectral del audio de Taylor Swift



Nota: En la figura se muestra la descomposición espectral del audio de Taylor Swift.

Las descomposiciones espectrales de cada audio se pueden ver en los anexos.

Cada audio, tanto los reales como los falsos, se dividió en ventanas de tiempo de 1 segundo y para cada ventana se extraen 27 características representativas de ese audio, tales como:

- **Chroma:** De acuerdo con (González Carrizo, 2017), es el promedio de la transformada de Fourier de corto plazo (STFT) en el espacio cromático. Representa la intensidad de las 12 clases de tonos en una escala musical y es útil para identificar patrones tonales y armonías en la señal de audio.
- **RMSE:** Es el promedio de la energía de la señal calculada con la raíz cuadrada media (RMSE). Esta mide la potencia de la señal en cada ventana y es útil para distinguir entre secciones de audio más suaves o enérgicas.
- **Spectral Centroid:** Promedio del centroide espectral, que indica la" gravedad" del espectro, es decir, la frecuencia media ponderada por la intensidad. Esta métrica ayuda a identificar si el sonido es agudo (alta frecuencia) o grave (baja frecuencia).
- **Spectral Bandwidth:** Es el promedio del ancho de banda espectral, que mide la

extensión de las frecuencias en torno al centroide espectral. Ayuda a diferenciar entre sonidos concentrados en una frecuencia o que se extienden a lo largo de varias.

- **Rolloff:** Es el promedio de la frecuencia de roll-off, que es la frecuencia por debajo de la cual se concentra un porcentaje significativo de la energía espectral, típicamente el 85%. Esto ayuda a distinguir entre sonidos con predominio de frecuencias bajas o altas.
- **Zero Crossing Rate:** Es el promedio de la tasa de cruce por cero, que cuenta la cantidad de veces que la señal cambia de signo (de positivo a negativo o viceversa) en cada ventana. Es útil para detectar sonidos percutivos o transitorios.
- **MFCC1 - MFCC20:** Son las medias de los coeficientes cepstrales de frecuencia melódica (MFCCs) del 1 al 20:

  Los MFCCs representan la envolvente del espectro de potencia de la señal en la escala mel. Cada coeficiente captura diferentes características del timbre y la forma de onda.

  - Los primeros coeficientes (como mfcc1 a mfcc5) suelen capturar las frecuencias de baja gama y son útiles para capturar los rasgos de la voz o timbre general.
  - Los coeficientes posteriores (mfcc6 a mfcc20) proporcionan detalles adicionales sobre las frecuencias medias y altas, que ayudan a diferenciar sonidos sutiles y característicos del hablante o de la fuente de sonido.

Además, tendremos nuestra variable objetivo que marca al audio como FAKE o REAL, esto en base a si sufrió una conversión de voz o no.

Preprocesamiento de datos

Partiendo del dataset crudo en el que se extrajeron las 27 características más importantes de los audios:

- Se hizo una normalización de datos para asegurar que todas las variables estén en la misma escala, esto dado que gran parte de los algoritmos que se van a implementar de ahora en adelante son sensibles a la escala de las características. De esta manera, se garantiza que todas las variables tengan la misma importancia y no exista un dominio de las escalas más grandes en los algoritmos.
- Se transforma en variable binaria la variable objetivo donde FAKE es 0 y REAL es 1.
- Se dividen los datos en datos de entrenamiento y de test en una proporción de 80:20, excepto para los algoritmos no supervisados.

Metodología usando Random Forest

Arquitectura del modelo

Para este algoritmo, se utilizaron los siguientes parámetros:

Número de árboles aleatorios: es la cantidad de árboles de decisión a entrenar para el modelo de clasificación de la variable objetivo. Para este caso, se usaron 500 árboles de decisión.

Estado aleatorio: este parámetro controla la reorganización de los datos antes de aplicar la división. Para este caso, se utiliza el valor de 42.

Profundidad máxima: limita la profundidad de los nodos de los árboles de decisión, lo que ayuda a prevenir el sobreajuste del modelo. Para este caso, se limitó a 8 niveles para cada árbol.

Criterio: especifica el criterio a utilizar para medir la calidad de la división de los nodos en un árbol. Para este caso, se usó el criterio" gini impurity"

Entrenamiento del modelo

Se utilizó el conjunto de datos de entrenamiento para que se ajustara al modelo creado con los parámetros descritos anteriormente, para después generar predicciones con el conjunto de datos de prueba.

Evaluación del modelo

Se utilizó el conjunto de datos de entrenamiento para que se ajustara al modelo creado con los parámetros descritos anteriormente, para después generar predicciones con el conjunto de datos de prueba.

Metodología usando Máquinas de Soporte Vectoriales (SVM)

Arquitectura del modelo

Para aplicar SVM's, primero se tuvo que reducir la dimensionalidad del dataset a dos componentes principales para visualización de los datos. Se aplicó el algoritmo PCA dentro del dataset obteniendo que las componentes obtenidas explican aproximadamente el 44% de la variabilidad de los datos originales. Esto significa que las dos componentes principales capturan alrededor de la mitad de la información total contenida en el dataset. Es posible que haya información importante en otras componentes que no se tomará en cuenta.

Luego de aplicar reducción de dimensionalidad, se dividió al conjunto de datos en datos de entrenamiento y prueba con una proporción 80:20 y se consideraron tres modelos SVM con diferentes kernels:

- Kernel RBF (Radial Basis Function).
- Kernel Polinomial con un grado de polinomio de 10.
- Kernel Sigmoidal con los siguientes parámetros:
- Gamma: afecta la forma de la frontera de decisión, mientras más alto, más se ajusta estrechamente a los datos de entrenamiento. Para este modelo, se usó un valor de 0.3.
- C: es un parámetro de regularización que controla la clasificación correcta de los puntos de datos de entrenamiento. Un valor de C alto intenta clasificar todos los puntos. Para este modelo, se usó un valor de 1.

Entrenamiento del modelo

Para cada modelo, se utilizó el conjunto de datos de entrenamiento para que se ajustara al modelo creado con los parámetros descritos anteriormente, para después generar predicciones con el conjunto de datos de prueba.

Adicional, se generó un afinamiento de parámetros mediante la librería de

GridSearchCV la cual nos permite encontrar los mejores parámetros para alcanzar un valor adecuado dentro de las métricas que se estén evaluando en cada modelo. Esto se realiza mediante el entrenamiento de los modelos con todas las combinaciones posibles de los parámetros por modificar.

Evaluación del modelo

De igual forma, se calcularon varias métricas como precisión, exactitud, sensibilidad, puntuación F1 y curva ROC, las cuales nos ayudan a corroborar el rendimiento del algoritmo y validar su implementación para el caso.

Implementación de una CNN

Para esta base de datos se usa la red CNN con capas Conv1D para procesar las características de las series de audio, estas se representan como una secuencia de datos unidimensionales por cada característica extraída. Normalmente, una capa Conv1D se usa para aplicar un kernel de convolución sobre una secuencia de datos dentro de una misma característica (como una serie temporal). Sin embargo, aquí se utiliza Conv1D para aprender patrones dentro de una secuencia de características espectrales de audio, lo que permite captar variaciones y patrones significativos en las diferentes variables espectrales (como el Chroma, RMSE, y los coeficientes MFCC). La Conv1D interpreta la relación entre los atributos de audio como una secuencia, lo que podría resaltar interacciones importantes entre las características espectrales.

El uso de una capa Conv1D en lugar de Conv2D en este caso se debe principalmente a la estructura de los datos. Conv1D es adecuada porque las características extraídas del audio están organizadas como una secuencia de valores unidimensionales por cada característica, y la red solo necesita capturar patrones dentro de esta secuencia de características, en lugar de relaciones bidimensionales como en una imagen.

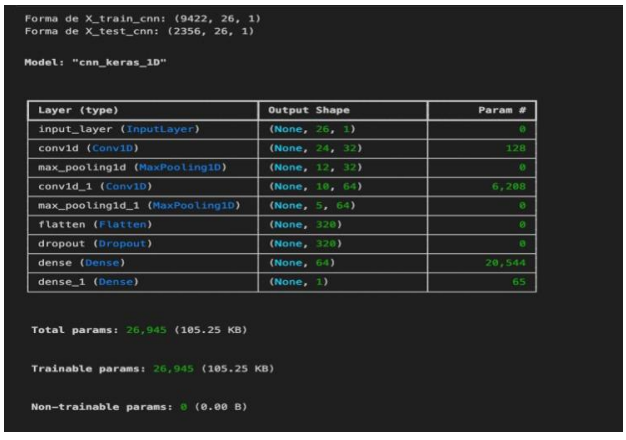
Arquitectura del modelo

Como se menciona en esta misma sección, se usa un modelo CNN basadas en capas Conv1D para procesar los datos de variables unidimensionales extraídas del audio (ver figura 6). A continuación se describen la arquitectura de la red neuronal:

1. **Capa de entrada (*input_layer*):** Este es el tamaño de las muestras de entrada, donde 26 es la longitud de la secuencia de características y 1 es el número de canales (características por entrada).
2. **Primera capa convolucional (*Conv1D*):** Aplica 32 filtros unidimensionales con un tamaño de kernel de 3 sobre la entrada. Esto reduce la longitud de la secuencia de 26 a 24 debido a la convolución.
3. **Primera capa de Pooling (*MaxPooling1D*):** Reduce la longitud de la secuencia a la mitad (de 24 a 12) aplicando un max pooling con tamaño de ventana 2. El número de filtros permanece igual.
4. **Segunda capa convolucional (*Conv1D*):** Aplica 64 filtros con un tamaño de kernel de 3. Reduce la longitud de la secuencia de 12 a 10.
5. **Capa de aplanamiento (*Flatten*):** Esta capa aplanada convierte la salida tridimensional en una secuencia unidimensional de tamaño 320 (5 x 64), que se conecta a las capas densas posteriores.
6. **Capa de Dropout (*Dropout*):** Aplica un mecanismo de regularización que omite aleatoriamente el 50% de las unidades durante el entrenamiento para evitar el sobreajuste.
7. **Capa de salida (*Dense*):** Es la capa de salida con una única neurona que utiliza la activación sigmoid para realizar la clasificación binaria.

Figura 6

Arquitectura de la red neuronal convolucional.



```
Forma de X_train_cnn: (9422, 26, 1)
Forma de X_test_cnn: (2356, 26, 1)

Model: "cnn_keras_1D"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 26, 1)	0
conv1d (Conv1D)	(None, 24, 32)	128
max_pooling1d (MaxPooling1D)	(None, 12, 32)	0
conv1d_1 (Conv1D)	(None, 10, 64)	6,208
max_pooling1d_1 (MaxPooling1D)	(None, 5, 64)	0
flatten (Flatten)	(None, 320)	0
dropout (Dropout)	(None, 320)	0
dense (Dense)	(None, 64)	20,544
dense_1 (Dense)	(None, 1)	65

```
Total params: 26,945 (105.25 KB)

Trainable params: 26,945 (105.25 KB)

Non-trainable params: 0 (0.00 B)
```

Nota: En la figura se muestra la red neuronal convolucional.

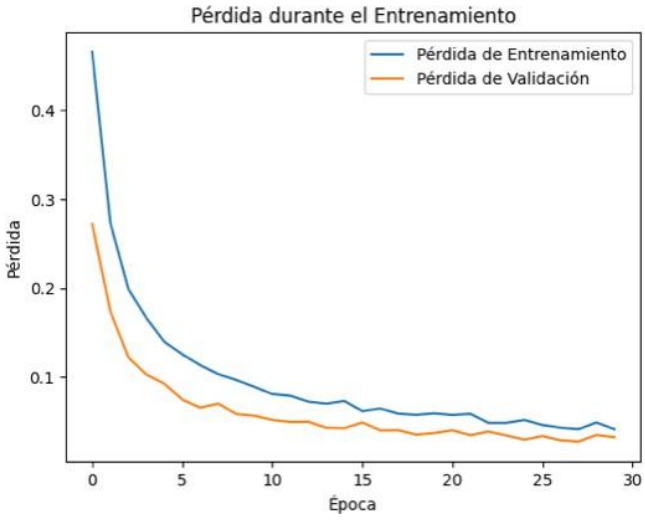
Entrenamiento y validación del modelo

Para la compilación se utilizó binary_crossentropy, ya que es un problema de clasificación binaria, el optimizado Adam ya que es un optimizador adaptativo bien establecido y como métrica de validación se utilizó el accuracy. El modelo fue entrenado durante 30 épocas con un batch size de 128. Se utilizó el conjunto de prueba para validar el rendimiento del modelo después de cada época. El entrenamiento fue monitoreado usando la precisión y la pérdida tanto para los datos de entrenamiento como de validación.

Las curvas de pérdida, figura 7, de entrenamiento y validación siguen trayectorias similares, lo que es una buena señal. Esto sugiere que el modelo no está sobreajustándose al conjunto de entrenamiento. Si la pérdida de validación comenzara a aumentar mientras la pérdida de entrenamiento disminuye, eso sería una señal de sobreajuste, pero en este caso, ambas curvas están alineadas.

Figura 7

Curvas de perdida de la red neuronal



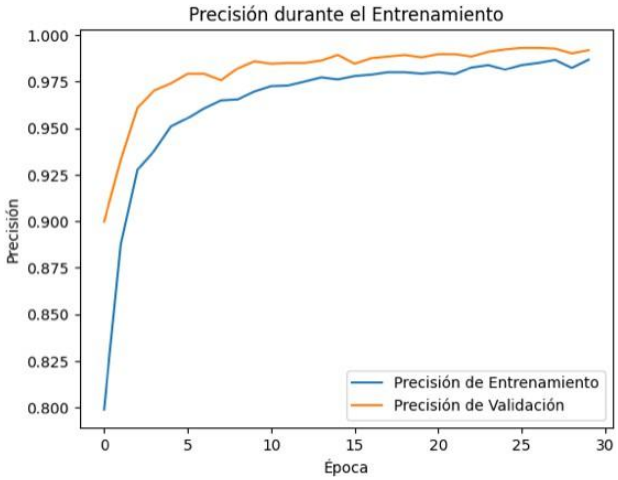
Nota: Curvas de perdida de la red neuronal convolucional en la data de

entrenamiento y testeo.

La precisión de validación se estabiliza alrededor de 0.98 (ver figura 8), lo cual indica que el modelo está realizando predicciones correctas la mayoría de las veces tanto en el conjunto de entrenamiento como en el de validación.

Figura 8

Curvas de precisión de la red neuronal



Nota: Curvas de precisión de la red neuronal convolucional en la data de entrenamiento y testeo.

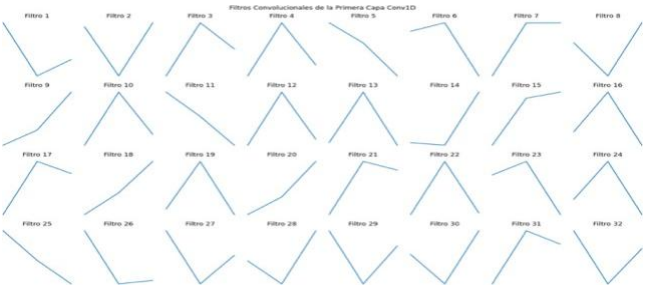
Además, se calcularon métricas como la precisión, exactitud, sensibilidad, puntuación F1 y AUC para evaluar el rendimiento del modelo.

Visualización de los filtros convolucionales de la primera capa del modelo Conv1D

Como se ve en la figura 9, los filtros convolucionales están capturando una variedad de patrones en los datos de entrada, lo que es esencial para que el modelo aprenda a distinguir entre audios "FAKE" y "REAL". Las formas angulares y las variaciones abruptas sugieren que el modelo está detectando diferentes transiciones en los datos de audio, como cambios en la frecuencia o intensidad.

Figura 9

Filtros convolucionales de la primera capa del modelo



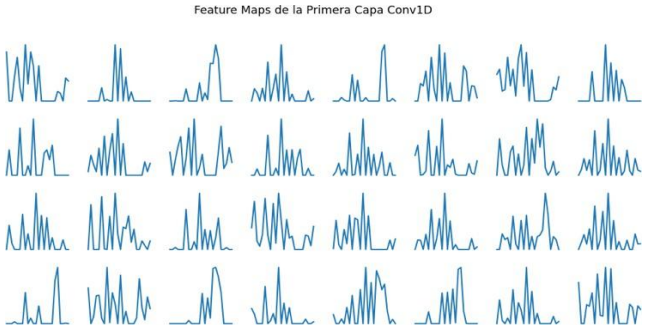
Nota: Filtros convolucionales de la primera capa del modelo CNN.

Visualización de los features map

Los feature maps muestran cómo la red neuronal está interpretando la entrada y qué partes de la señal son más relevantes para cada filtro (ver figura 10). Los filtros están capturando picos, transiciones y variaciones que pueden ser indicativas de si el audio es FAKE o REAL.

Figura 10

Feature maps del modelo CNN.



Nota: Features maps que muestran la interpretación de la entrada para cada filtro de la red neuronal convolucional.

Implementación de modelos de aprendizaje semi-supervisado

Para aplicar los modelos de aprendizaje semi-supervisado es necesario simular un escenario de datos parcialmente etiquetados, eliminando aleatoriamente el 20% de las etiquetas (FAKE o REAL) y reemplazándolas con −1, representando datos sin etiquetar.

Preprocesamiento para los modelo de aprendizaje semi-supervisado

Se simula un escenario de datos parcialmente etiquetados, eliminando aleatoriamente el 20% de las etiquetas (FAKE o REAL) y reemplazándolas con -1, representando datos sin etiquetar.

Self Training

Para esta implementación se utiliza un Self-Training Classifier de scikit-learn, que envuelve un RandomForestClassifier como modelo base.

Funcionamiento del Self-Training:

Entrenamiento inicial: Se entrena un modelo base (en este caso, un RandomForestClassifier) con los datos etiquetados disponibles.

Pseudo-etiquetado: El modelo entrenado se utiliza para predecir las etiquetas de los datos no etiquetados.

Selección de muestras: Se seleccionan las predicciones con mayor confianza (por encima de un umbral definido) y se agregan al conjunto de datos de entrenamiento como pseudo-etiquetas.

Reentrenamiento: El modelo base se vuelve a entrenar con el conjunto de datos aumentado (datos etiquetados originales + pseudo-etiquetas).

Iteración: Los pasos 2-4 se repiten hasta que se cumple un criterio de parada (por ejemplo, un número máximo de iteraciones).

Entrenamiento:

1. El Self-Training Classifier se entrena con los datos de entrenamiento.
2. Se evalúa el modelo utilizando el conjunto de prueba y la métrica de accuracy, que mide el porcentaje de predicciones correctas.
3. Se realiza una búsqueda de hiperparámetros con GridSearchCV, optimizando el número de estimadores, la profundidad máxima del RandomForest y el umbral de confianza del Self-Training Classifier, configurando la siguiente grilla para obtener los mejores hiperparámetros:

- base_estimator_n_estimators: [10, 50, 100],
- base_estimator_max_depth: [None, 5, 10],
- threshold: [0.8, 0.7, 0.9]

Validación:

- La métrica principal utilizada es accuracy obteniendo un valor de 0.985.
- Los mejores hiperparámetros encontrados por GridSearchCV y la mejor puntuación (accuracy) alcanzada durante la validación cruzada son los siguientes.
  base_estimator_max_depth: None
  base_estimator_n_estimators: 100
  threshold: 0.8

Label Propagation

Para esta implementación se utiliza LabelPropagation de scikit-learn, que envuelve un KNN como modelo base.

Funcionamiento de Label Propagation

Entrenamiento inicial: Se entrena un modelo inicial de Label Propagation con los datos etiquetados disponibles. Se utiliza el kernel 'knn' para definir la relación entre los puntos de datos.

Pseudo-etiquetado: El modelo entrenado se utiliza para predecir las etiquetas de los datos no etiquetados (aquellos con etiqueta −1). Estas predicciones se consideran "pseudo-etiquetas".

Selección de muestras: En este caso, la selección de muestras se realiza de forma implícita por el algoritmo Label Propagation. El algoritmo considera la confianza en las pseudo-etiquetas al propagar las etiquetas a través del grafo de datos.

Reentrenamiento: El modelo se reentrena con los datos etiquetados originales y los datos pseudo-etiquetados.

Iteración: Los pasos 2, 3 y 4 se repiten iterativamente hasta que el modelo converge o se alcanza un número máximo de iteraciones. En este caso, el número de iteraciones está definido implícitamente por la clase Label Propagation.

Entrenamiento:

1. Se utiliza la clase Label Propagation de scikit-learn para crear el modelo de self-learning.
2. Se realiza una búsqueda de hiperparámetros utilizando GridSearchCV para encontrar la mejor combinación de n_neighbors y gamma para el kernel 'knn', utilizando la siguiente grilla de hiperparámetros:
   kernel: ['knn']
   n_neighbors: [3, 5, 7, 9]
   gamma: [0.05, 0.1, 0.5, 1.0]
3. Se evalúa el modelo con los mejores hiperparámetros en el conjunto de prueba.

Validación:

- La métrica principal utilizada es accuracy obteniendo un valor de 0.803
- Los mejores hiperparámetros encontrados por GridSearchCV y la mejor puntuación (accuracy) alcanzada durante la validación cruzada son los siguientes.
  kernel: ['knn']
  n_neighbors: 5
  gamma: 0.05

Label Spreading

Se utiliza un LabelSpreading de scikit-learn, que envuelve un KNN y RBF como modelos base

Funcionamiento de Label Spreading

Construcción del grafo de similitud: Se crea un grafo donde cada punto de datos es un nodo y las aristas conectan nodos con características similares. La similitud se define mediante el kernel especificado (por ejemplo, 'knn' o 'rbf').

Inicialización de etiquetas: Los puntos de datos etiquetados conservan sus etiquetas originales. Los puntos sin etiquetar (con etiqueta -1) se inicializan con una distribución uniforme de probabilidades para cada clase.

Propagación iterativa de etiquetas:

- Se actualizan las etiquetas de los puntos sin etiquetar basándose en las etiquetas de sus vecinos en el grafo.
- La influencia de los vecinos se pondera por la similitud entre ellos.
- Este proceso se repite iterativamente, permitiendo que las etiquetas se propaguen a través del grafo.

Convergencia: El algoritmo continúa iterando hasta que las etiquetas de los puntos sin etiquetar convergen o se alcanza un número máximo de iteraciones.

Predicción: Una vez que el algoritmo converge, las etiquetas finales de los puntos sin etiquetar se utilizan como predicciones.

Entrenamiento:

1. Se utiliza la clase LabelSpreading de scikit-learn para crear el modelo de self-learning.
2. Se realiza una búsqueda de hiperparámetros utilizando GridSearchCV para encontrar la mejor combinación de nneighbors y gamma para el kernel 'knn' y 'rbf', utilizando la siguiente grilla de hiperparámetros:
   kernel: ['knn', 'rbf']
   n_neighbors: [3, 5, 7, 9]
   gamma: [0.05, 0.1, 0.5, 1.0]
3. Se evalúa el modelo con los mejores hiperparámetros en el conjunto de prueba.

Validación:

- La métrica principal utilizada es accuracy obteniendo un valor de 0.809.
- Los mejores hiperparámetros encontrados por GridSearchCV y la mejor puntuación (accuracy) alcanzada durante la validación cruzada son los siguientes.
  kernel: ['knn']
  n_neighbors: 5
  gamma: 0.05

Metodología usando K-means Clustering

Preprocesamiento

Eliminación de la Etiqueta: Como estamos trabajando con aprendizaje no supervisado, las etiquetas (”FAKE” o ”REAL”) no son necesarias para el proceso de clustering. Por lo tanto, eliminamos la columna de etiquetas del conjunto de datos.

División de los Datos: A diferencia de los modelos supervisados, en el aprendizaje no supervisado no necesitamos dividir los datos en conjuntos de entrenamiento y prueba. Los datos completos se usan para la tarea de clustering.

Arquitectura del Modelo

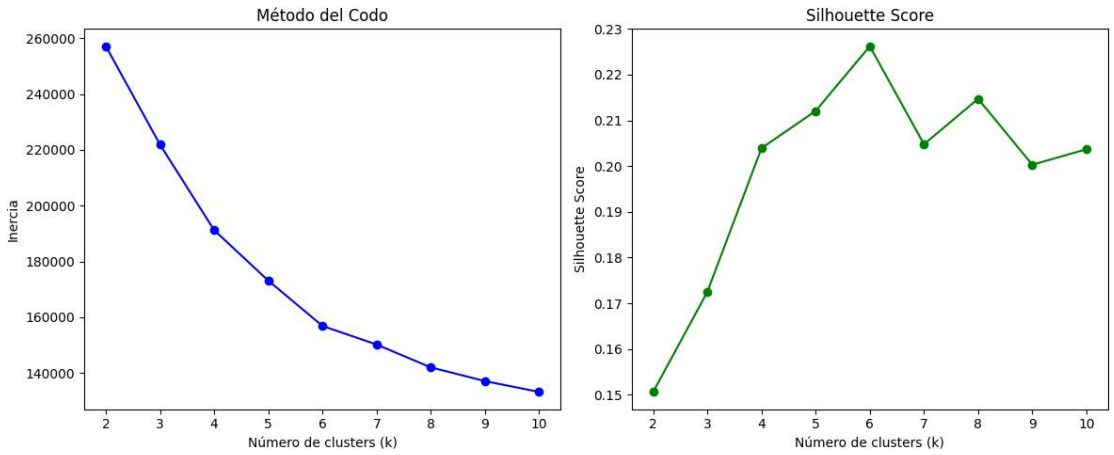
Se utilizará el algoritmo K-means para realizar el agrupamiento en los datos. K-means es un algoritmo de agrupamiento que intenta encontrar un número específico de grupos (o clusters) en los datos, minimizando la distancia dentro de cada grupo.

El modelo de K-means no sigue una arquitectura de red neuronal típica, ya que es un algoritmo de clustering. El modelo trabaja agrupando datos en K clústeres, basado en la similitud de sus características. Cada punto de datos se asigna al clúster cuyo centroide está más cercano.

1. **Inicialización:** Se elige un número de clusters (*k*) que representa el número de grupos que se desean encontrar en los datos. Para esto se han elegido 6, obtenidos mediante el gráfico.

Figura 11

Método del codo y silhoutte score



Nota: Gráfico del Método del Codo y Silouette Score para determinar el número adecuado de clusters.

2. **Asignación de puntos:** Internamente, el algoritmo calcula las distancias entre cada punto y los centroides actuales y asigna cada punto al centroide más cercano.
3. **Actualización de centroides:** Los centroides de los clusters se recalculan en función de los puntos asignados a cada uno. Después de asignar los puntos, el algoritmo recalcula la posición de cada centroide como el promedio de las coordenadas de los puntos asignados a ese cluster.
4. **Iteración:** Los pasos 2 y 3 se repiten hasta que los centroides ya no cambian significativamente o se alcanza un número máximo de iteraciones.

Entrenamiento

El modelo de clustering K-Means fue entrenado utilizando los datos escalados, con un número de clusters establecido en 6. Durante el entrenamiento, el algoritmo comenzó con la inicialización de 6 centroides seleccionados aleatoriamente, asegurando la reproducibilidad mediante el parámetro random_state=42. En cada iteración, los puntos del dataset fueron asignados al cluster correspondiente al centroide más cercano, utilizando la métrica de distancia euclidiana. Posteriormente, los centroides fueron recalculados como el promedio de las posiciones de los puntos asignados a cada cluster.

Este proceso de asignación de puntos y actualización de centroides continuó de manera iterativa hasta que los cambios en las posiciones de los centroides fueron insignificantes o se alcanzó el criterio de convergencia basado en la minimización de la inercia, que corresponde a la suma de las distancias cuadradas entre los puntos y sus centroides asignados.

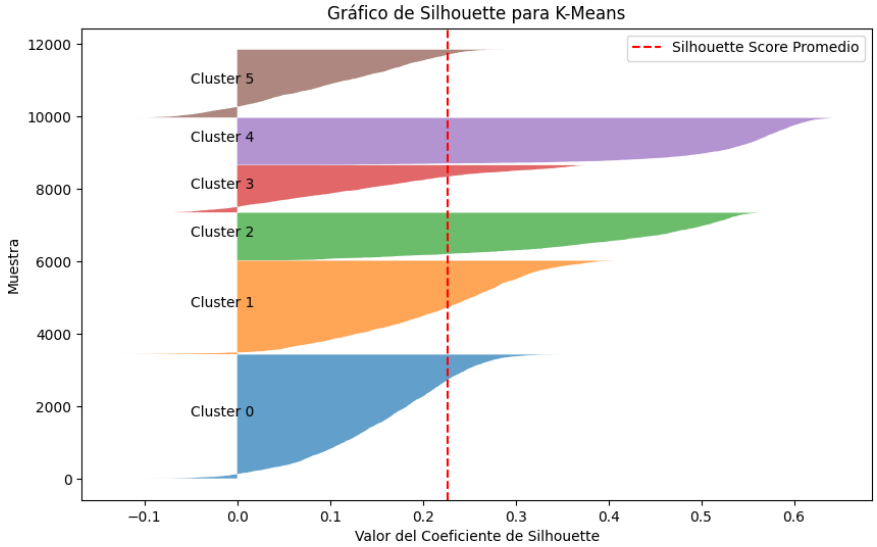
Evaluación del modelo

Para evaluar la calidad del modelo, se calculó el Silhouette Score. Además, para facilitar la interpretación y visualización de los clusters, se utilizó el análisis de componentes principales (PCA) para reducir las dimensiones del dataset a dos componentes

principales, permitiendo representar gráficamente los clusters generados en un plano bidimensional.

Figura 12

Silhouette Score



Nota: EL gráfico muestra mediante una línea interpuntada el valor de Silhouette Score para la evaluación del modelo.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Preprocesamiento

Eliminación de la Etiqueta: Como estamos trabajando con aprendizaje no supervisado, las etiquetas (”FAKE” o ”REAL”) no son necesarias para el proceso de clustering. Por lo tanto, eliminamos la columna de etiquetas del conjunto de datos.

División de los Datos: A diferencia de los modelos supervisados, en el aprendizaje no supervisado no necesitamos dividir los datos en conjuntos de entrenamiento y prueba.

Arquitectura del Modelo

Para realizar el agrupamiento en los datos, se emplea el algoritmo DBSCAN

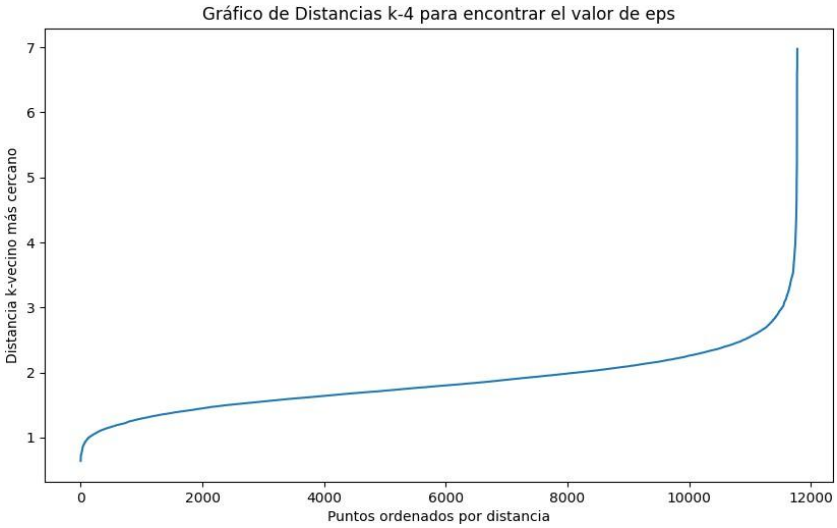
(Density-Based Spatial Clustering of Applications with Noise), que es un algoritmo de clustering basado en la densidad. DBSCAN identifica clústeres de puntos densos, separándolos de las regiones de baja densidad (ruido).

El modelo de DBSCAN no sigue una arquitectura de red neuronal, ya que es un algoritmo de agrupamiento basado en la densidad. El modelo agrupa los datos en función de la densidad de puntos cercanos y asigna aquellos puntos que no pertenecen a ningún clúster como ruido.

- **Inicialización:** Se definen dos parámetros principales:
- **eps:** La distancia máxima entre dos puntos para que se consideren vecinos. En este caso, se ha fijado en 2.5, un valor obtenido tras realizar un análisis gráfico de distancias de los k-vecinos más cercanos.
- **min_samples:** El número mínimo de puntos necesarios para formar un clúster denso. En este caso, como se puede ver en la figura 13, se ha fijado en 5.
- **Asignación de puntos:** El algoritmo selecciona un punto al azar y busca todos los puntos dentro de su vecindad (con distancia menor que **eps**). Si el número de puntos en esa vecindad es mayor o igual a **min_samples**, esos puntos forman un clúster. Si no, el punto se marca como ruido.
- **Expansión del clúster:** Si un punto está dentro de un clúster, los puntos cercanos dentro de **eps** también se consideran parte del clúster y se repite el proceso hasta que no se encuentren más puntos cercanos.
- **Manejo del ruido:** Los puntos que no cumplen con los criterios de densidad son etiquetados como ruido (etiquetados como -1). Al final del proceso, el modelo devuelve los clústeres encontrados y los puntos que son considerados ruido. El número total de clústeres se calcula restando los puntos de ruido de los posibles clústeres encontrados.

Figura 13

Gráfico de Distancias para encontrar el valor de eps



Nota: Gráfico de las distancias de los k-vecinos más cercanos que ayuda a determinar el valor óptimo de eps.

Entrenamiento

El modelo de clustering DBSCAN fue entrenado utilizando los datos escalados, con los parámetros eps y min_samples establecidos en 2.5 y 5, respectivamente. Durante el entrenamiento, el algoritmo comenzó por seleccionar un punto al azar del dataset y evaluó su vecindad para determinar si formaba un clúster denso, es decir, si el número de puntos dentro de la distancia eps era mayor o igual a min_samples.

En cada iteración, DBSCAN asignó a los puntos cercanos dentro del umbral eps al clúster correspondiente si cumplían con el criterio de densidad. Los puntos que no cumplían con los criterios de densidad fueron etiquetados como ruido (etiquetados como -1). Este proceso de asignación de puntos y expansión del clúster se repitió hasta que no se encontraron más puntos cercanos para ampliar los clústeres existentes.

A medida que el algoritmo avanzaba, los puntos que inicialmente estaban fuera de los clústeres, pero que eran cercanos a otros puntos densos, fueron gradualmente asignados a esos clústeres, extendiendo las regiones densas.

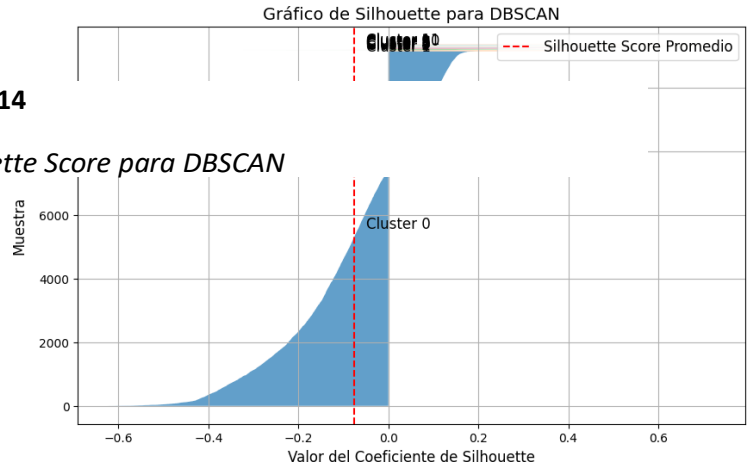
El proceso continuó de manera iterativa hasta que todos los puntos fueron evaluados y asignados a un clúster o etiquetados como ruido. El número total de clústeres se calculó restando los puntos etiquetados como ruido.

Evaluación del modelo

El modelo se evalúa utilizando el Silhouette Score, que mide qué tan bien se ha logrado el agrupamiento, considerando tanto la cohesión dentro del clúster como la separación entre clústeres. Este valor se calcula sólo para los puntos que no son ruido.

Figura 14

Silhouette Score para DBSCAN



Nota: EL gráfico muestra mediante una línea interpuntada el valor de Silhouette Score para la evaluación del modelo.

Autoencoders

Preprocesamiento

Eliminación de la Etiqueta: Como estamos trabajando con aprendizaje no supervisado, las etiquetas (”FAKE” o ”REAL”) no son necesarias para el proceso de clustering. Por lo tanto, eliminamos la columna de etiquetas del conjunto de datos.

Normalización de los Datos: Para mejorar el rendimiento del autoencoder, normalizamos los datos de entrada para asegurarnos de que todos los valores estén en un rango similar.

Arquitectura del Modelo

El modelo implementado se basa en un Autoencoder, una red neuronal diseñada para aprender una representación comprimida (codificada) de los datos de entrada, logrando una reconstrucción aproximada de estos mismos datos. Este enfoque es útil para la detección de anomalías, dado que las observaciones anómalas suelen tener un error de reconstrucción mayor en comparación con los datos normales.

1. **Entrada:** La red neuronal recibe como entrada un conjunto de datos numéricos

normalizados con una dimensión de entrada de *input_dim*, equivalente al número de características en el conjunto de datos original.

2. **Capa de codificación:**

   Es una capa densa (*Dense Layer*) con *encoding_dim* = 32 nodos y una función de activación *ReLU*.

   Esta capa aprende una representación comprimida de los datos, reduciendo la dimensionalidad y extrayendo las características más relevantes.

3. **Capa de decodificación:**

   Es una capa densa (*Dense Layer*) con el mismo número de nodos que la entrada (*input_dim*) y una función de activación *sigmoid*.

   Esta capa reconstruye los datos originales a partir de la representación comprimida generada en la capa de codificación.

4. **Optimización:**

   Se emplea el optimizador *Adam* para minimizar la función de pérdida *mean squared error (MSE)*.

   El entrenamiento se realiza durante 50 épocas con un tamaño de lote de 256, utilizando el mismo conjunto de datos de entrada para entrenamiento y validación, dada la naturaleza no supervisada del modelo.

5. **Detección de anomalías:**

   Se calcula el error de reconstrucción para cada muestra como la diferencia cuadrática media (*MSE*) entre los datos originales y los datos reconstruidos.

   Los datos con errores de reconstrucción superiores al percentil 95 se consideran anomalías.

Además, para analizar la calidad de las representaciones comprimidas generadas por el Autoencoder, se utiliza el algoritmo K-Means para agrupar los datos reconstruidos. Posteriormente, se calcula el Silhouette Score para evaluar la cohesión y separación de los clústeres formados, obteniendo un puntaje de silhouette_score_autoencoder, que mide la calidad del agrupamiento en el espacio latente aprendido por el modelo.

Entrenamiento

El modelo **Autoencoder** fue entrenado utilizando los datos normalizados mediante la técnica de *StandardScaler*, asegurando que todas las variables tuvieran una media de 0 y una desviación estándar de 1. Este proceso de normalización es esencial para garantizar la

estabilidad del entrenamiento en redes neuronales y evitar problemas derivados de escalas dispares entre las características del dataset.

El modelo fue configurado con una dimensión de entrada igual al número de características del dataset (input_dim), una capa de codificación con 32 nodos (encoding_dim) y una capa de decodificación con input_dim nodos. En la capa de codificación se utilizó la función de activación ReLU, mientras que en la capa de decodificación se empleó sigmoid, lo que permitió generar reconstrucciones que se mantuvieran dentro del rango de los valores originales normalizados. La compilación del modelo se realizó utilizando el optimizador Adam con una tasa de aprendizaje predeterminada, y la función de pérdida seleccionada fue el mean squared error (MSE), adecuada para capturar diferencias entre los datos originales y sus reconstrucciones.

El modelo fue entrenado durante 50 épocas con un tamaño de lote de 256 muestras por iteración. Este proceso incluyó la división interna del conjunto de datos en particiones de entrenamiento y validación, lo que permitió monitorear el desempeño del modelo y el error de reconstrucción en cada época. Al final del entrenamiento, el modelo generó reconstrucciones de las muestras originales, calculándose el error de reconstrucción como el MSE promedio entre los datos originales y sus reconstrucciones.

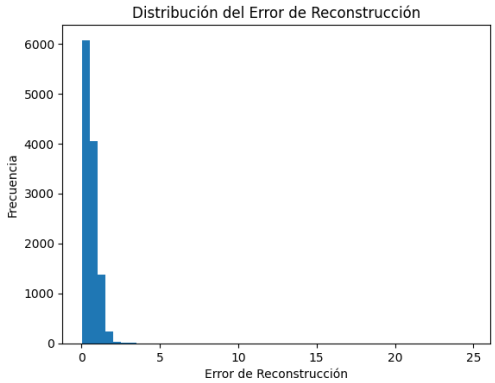
Para identificar posibles anomalías en los datos, se utilizó la distribución del error de reconstrucción. Aquellas muestras cuyo error superaba el percentil 95 de la distribución fueron marcadas como anomalías. Este enfoque permitió distinguir entre datos que el modelo pudo reconstruir correctamente y aquellos que se desviaban significativamente de los patrones generales del dataset, indicando su posible naturaleza anómala.

Evaluación del modelo

Error de reconstrucción: La principal métrica para evaluar el rendimiento de un autoencoder es el error de reconstrucción, que mide la diferencia entre las entradas originales y las salidas reconstruidas. Un valor bajo indica que el autoencoder ha aprendido una representación eficiente de los datos.

Figura 15

Error de Reconstrucción



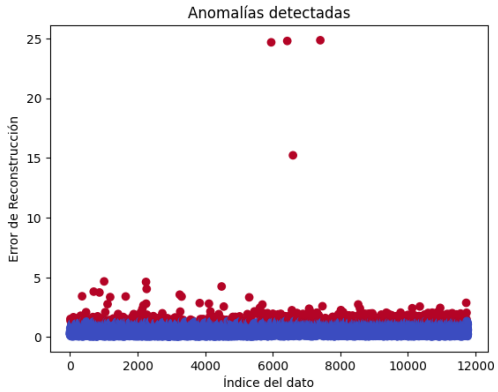
Nota: Este gráfico muestra la distribución de los valores de error de reconstrucción calculados para cada punto del dataset.

Visualización de la representación latente: Se puede analizar visualmente, ver figura 16, la representación latente generada por el encoder, para verificar si los datos se agrupan correctamente y si las características importantes están siendo capturadas.

Silhouette Score: El Silhouette Score es una métrica que evalúa qué tan bien están definidos los clústeres, ver figuta 12. Su valor varía entre -1 y 1.

Figura 16

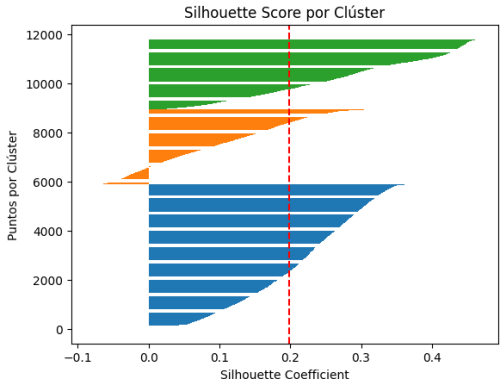
Anomalías Detectadas



Nota: Este gráfico visualiza el error de reconstrucción de cada punto en el dataset, ordenados por índice en el eje X.

Figura 17

Silhouette Score



Nota: Este gráfico muestra la distribución del Silhouette Score para cada punto dentro de los clústeres, proporcionando una vista más detallada.

Evaluación de Modelos de Clustering

Para comparar el desempeño de los modelos de clustering, se utiliza el Silhouette Score, que mide la calidad de los clusters formados. El Silhouette Score combina dos factores clave: a, que mide la cohesión dentro del cluster (distancia promedio entre un punto y los puntos de su propio cluster), y b, que mide la separación (distancia promedio entre un punto y los puntos del cluster más cercano). La fórmula es la siguiente:

$$s = \frac{b - a}{\max(a, b)}$$

Donde:

a: Distancia promedio entre un punto y los puntos de su propio cluster.

b: Distancia promedio entre un punto y los puntos del cluster más cercano.

El valor de s está en el rango $[-1, 1]$, donde valores cercanos a 1 indican clusters bien definidos, valores cercanos a 0 indican solapamiento de clusters, y valores negativos sugieren asignaciones incorrectas.

Modelo K-means

El algoritmo K-means divide los datos en k clusters, minimizando la suma de las distancias cuadráticas de los puntos a sus respectivos centroides. Su evaluación con el Silhouette Score se realiza sobre los clusters generados. Un puntaje alto indica que los puntos están bien agrupados alrededor de sus centroides y que los clusters están claramente

separados.

Ventajas:

- Fácil de implementar.
- Eficiente computacionalmente.

Limitaciones:

- Depende del número de clusters k que se define previamente.
- No es robusto a clusters de forma irregular o ruido.

Modelo DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) agrupa puntos en función de su densidad, identificando clusters de forma arbitraria y diferenciando puntos ruidosos como outliers. Para evaluarlo con el Silhouette Score, solo se consideran los puntos asignados a clusters válidos.

Ventajas:

- Identifica clusters de forma arbitraria.
- Maneja ruido y outliers.

Limitaciones:

- Sensible a los parámetros ϵ (radio) y minPts (mínimo número de puntos en el vecindario).
- En conjuntos de datos con densidad variable, puede no generar clusters claros.

Modelo Basado en Autoencoders

Los autoencoders, modelos de redes neuronales no supervisados, se utilizan para reducir la dimensionalidad y extraer características relevantes antes de aplicar clustering. Tras el preprocesamiento, se puede aplicar un algoritmo como K-means y evaluar el resultado con el Silhouette Score.

Ventajas:

- Capturan relaciones no lineales en los datos.

- Mejoran la calidad del clustering en datos complejos.

Limitaciones:

- Requieren mayor poder computacional.
- Pueden sobreajustarse si no se regularizan adecuadamente.

Comparación de Resultados

Tras calcular el Silhouette Score para cada modelo, el desempeño de los algoritmos se compara directamente. El modelo con el mayor Silhouette Score es considerado el más efectivo en agrupar los datos de forma coherente y separada.

$$\text{Modelo ganador: } \arg\max_{s_{\text{modelo}}} (s_{\text{kmeans}}, s_{\text{dbscan}}, s_{\text{autoencoder}})$$

Capítulo 3

Resultados

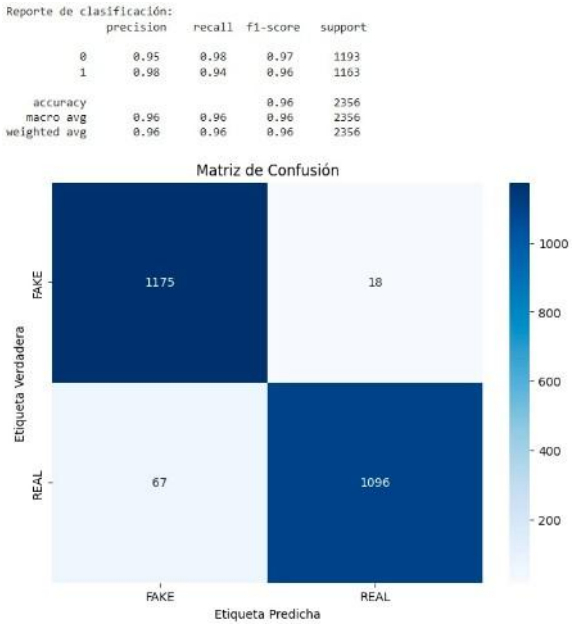
Random Forest

La precisión del modelo ronda aproximadamente un 98%, lo cual indica que el modelo está realizando las predicciones correctas en su mayoría, de acuerdo con sus conjuntos de datos, tanto de entrenamiento como de prueba.

En la figura 18 se presenta el reporte de clasificación y la matriz de confusión del modelo entrenado. Se observa que los valores de precisión y recall, así como el f1-score son cercanos a uno, lo cual evidencia la gran efectividad que tiene el modelo para la predicción de un audio real o falso.

Figura 18

Matriz de confusión para algoritmo Random Forest

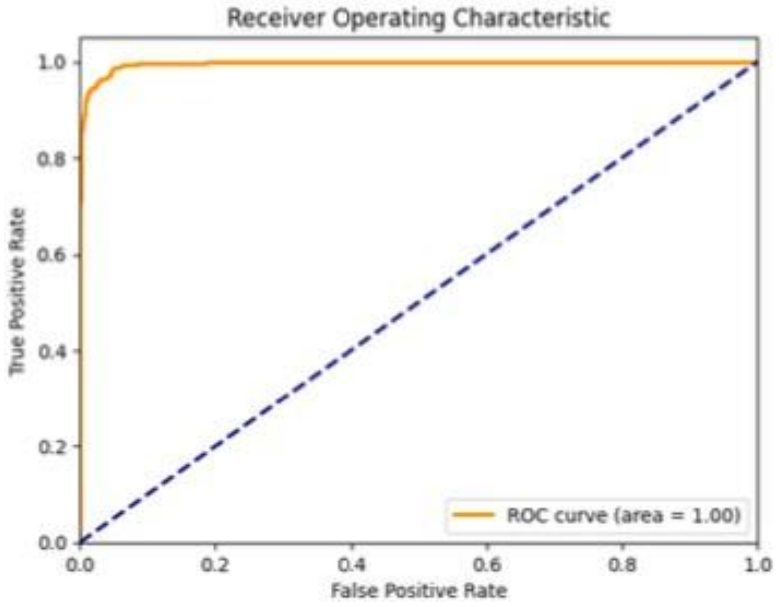


Nota: La matriz de confusión ayuda a verificar las predicciones correctas y erróneas realizadas por el modelo, de manera visual.

Adicional, se calculó la curva ROC como parte de la evaluación del modelo, en donde se puede observar en la figura 19 que la curva se cierra más en la zona superior izquierda, maximizando la tasa de valores positivos reales y minimizando la tasa de valores de falsos positivos. Esto indica que el rendimiento del modelo es el adecuado para la predicción.

Figura 19

Curva ROC algorimo Random Forest



Nota: La curva ROC mide la compensación que existe entre valores verdaderos positivos y negativos en umbrales diferentes.

A manera de resumen, se presentan las métricas calculadas para este modelo en la tabla 2.

Tabla 2

Métricas de evaluación para Random Forest

Métrica	Valor
AUC	1.00
Accuracy	0.96
Precisión	0.94

Recall	0.94
F1 Score	0.96

Nota: Esta tabla presenta los resultados del modelo Random Forest.

Máquinas de Soporte Vectorial

Luego del entrenamiento y predicción de los modelos, se obtuvieron los siguientes valores de precisión, descritos en la tabla 3.

Tabla 3

Precisión de modelos SVM de acuerdo con las funciones Kernel

RBF	**Polinomial**	**Sigmoidal**
0.738	0.652	0.444

Nota: Esta tabla presenta los resultados del modelo SVM con diferentes Kernel.

Se observa que el modelo con mejor precisión obtenida es el modelo SVM con kernel 'RBF' alcanzando un 74% de precisión aproximadamente. Sin embargo, este valor sigue siendo bajo para considerar a dicho modelo como válido para predicciones. Es necesario ajustar parámetros para lograr mayor exactitud.

Por lo tanto, se realizó un ajuste de parámetros mediante la librería de GridSearchCV para obtener una mejor precisión para el modelo. En la tabla 4 se presentan los resultados del afinamiento de parámetros. Cabe recalcar que los parámetros por ajustar fueron "C" y "gamma", y las funciones kernel escogidas para el modelo fueron "RBF" y "Sigmoidal".

Tabla 4

Resultados de afinamiento de parámetros para SVM

	Resultados		
	C	Gamma	Precisión
RBF	100	1	0.77
Sigmoide	1000	0.0001	0.62

Nota: Esta tabla presenta los resultados del GridSearch del modelo SVM.

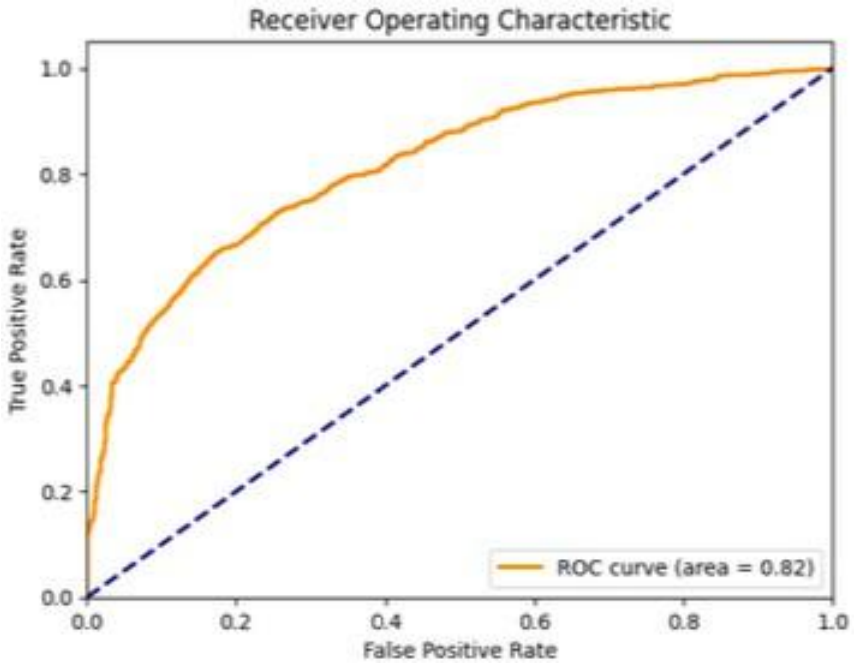
Se obtuvo mejorías con respecto a los primeros entrenamientos; no obstante, siguen siendo unos valores de precisión bajos para considerar estos modelos como óptimos para la

clasificación de DeepFake de audios. Analizando los resultados, se optó por escoger el modelo SVM con función kernel “RBF” para el cálculo de la curva ROC, debido a la precisión que genera, la cual es la mejor de entre las funciones kernel implementadas.

Finalmente, en la figura 20 se observa la curva ROC generada con el modelo escogido. El AUC calculado es de 0.82. Es un valor cercano a 1, pero no es factible su rendimiento. Puede clasificar de manera incorrecta los audios que procese. Esto se representa en la curva, ya que su curvatura no se acerca a la esquina superior izquierda donde el rendimiento del modelo es el adecuado. No obstante, tiene un rendimiento aceptable.

Figura 20

Curva ROC algoritmo SVM



Nota: Se observa la curvatura alejada de la esquina superior izquierda. Esto indicado un rendimiento regular del modelo.

A manera de resumen, se presentan las métricas calculadas para este modelo en la tabla 5.

Tabla 5

Métricas de evaluación para Máquinas de Soporte Vectorial

Métrica	Valor
AUC	0.82
Accuracy	0.77
Precisión	0.79
Recall	0.64
F1 Score	0.71

Nota: Esta tabla presenta los resultados del modelo SVM

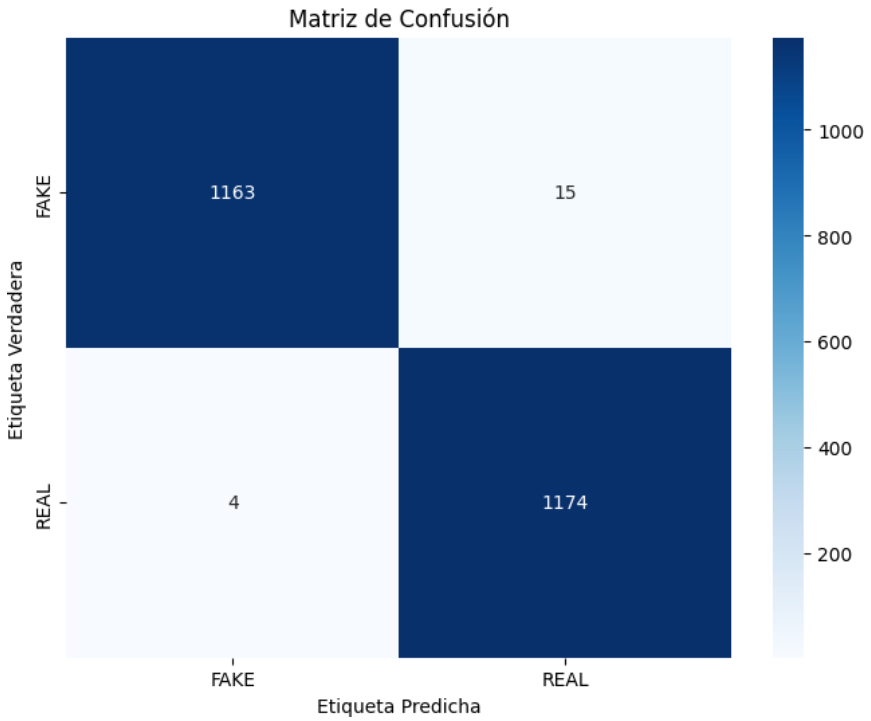
Redes neuronales convolucionales

La precisión del modelo es aproximadamente de 99%, lo que indica que el modelo tiene un buen poder predictivo, además validamos que no existe sobreajuste al obtener validar sobre el conjunto de testeo.

En la figura 21 se presenta la matriz de confusión del modelo cnn y se observa que los valores de precisión y recall, así como el f1-score son cercanos a uno, lo cual indica el buen comportamiento del modelo que al clasificar audios reales y falsos.

Figura 21

Matriz de confusión para algoritmo CNN

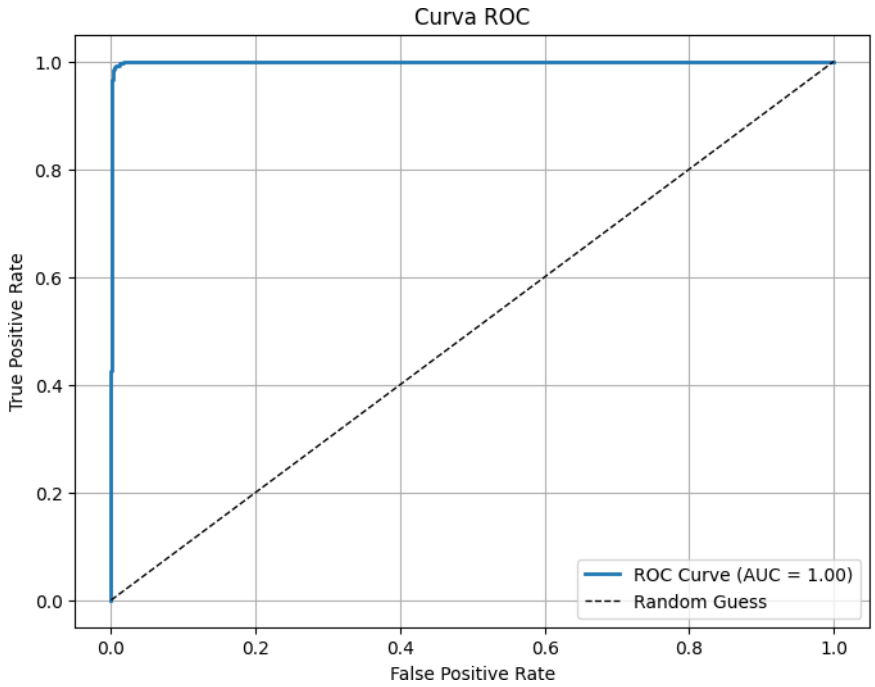


Nota: La matriz de confusión nos indica que el modelo está clasificando correctamente.

En la figura 22 se presenta la curva ROC del modelo CNN, se observa que el modelo tiene un alto rendimiento predictivo pues la curva se acerca mucho al borde superior izquierdo.

Figura 22

Curva ROC algorimo CNN



Nota: La curva ROC señala un alto rendimiento del modelo de redes neuronales convolucionales.

Nuevamente, a manera de resumen, se presentan las métricas calculadas para el modelo CNN en la tabla 6.

Tabla 6

Métricas de evaluación para CNN.

Métrica	Valor
AUC	1.00
Accuracy	0.99

Precisión	0.99
Recall	0.99
F1 Score	0.99

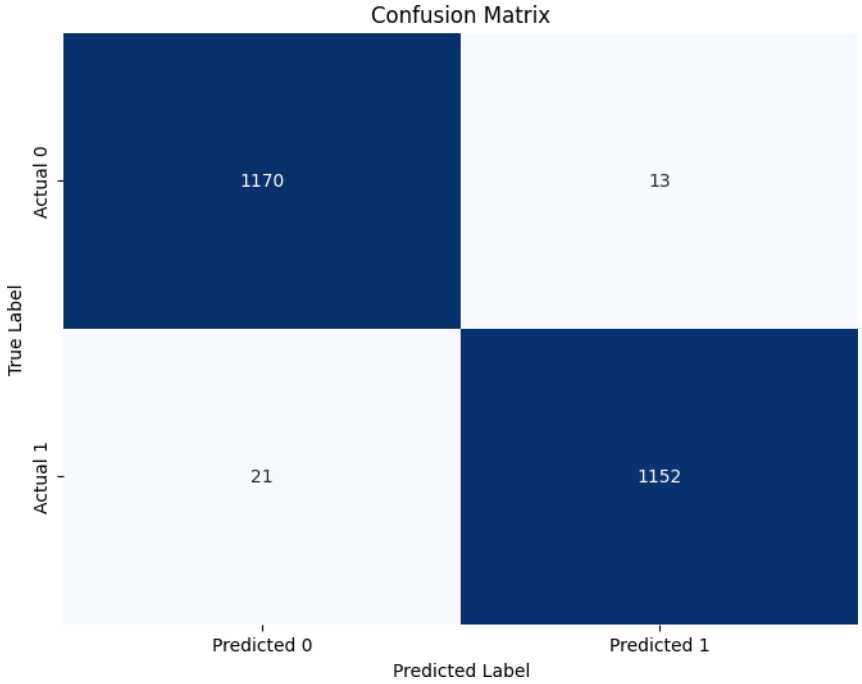
Nota: Esta tabla presenta los resultados del modelo CNN

Self Training

En la figura 23 se presenta la matriz de confusión del modelo Self Training, se observa que si bien el modelo no es malo, se equivoca al predecir la clase 1 (que un audio sea FALSO).

Figura 23

Matriz de confusión para algoritmo Self Training

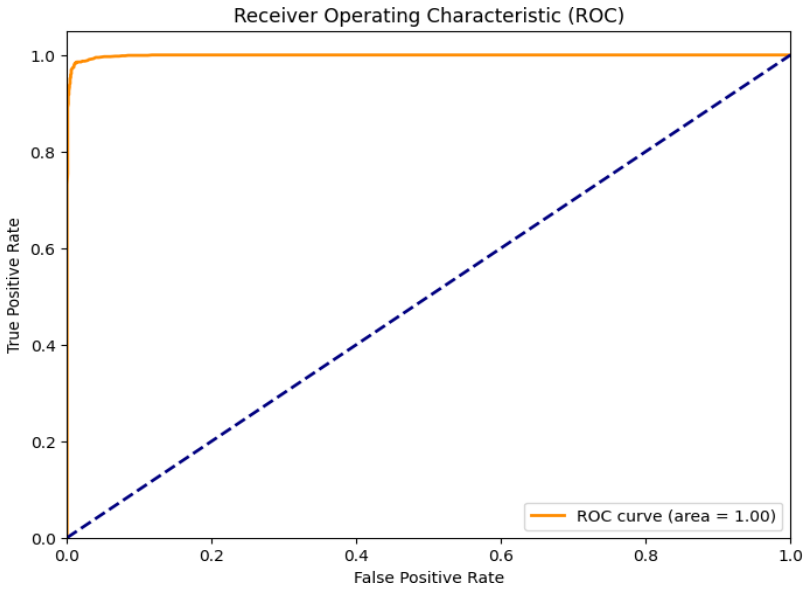


Nota: La matriz de confusión indica que el algoritmo está cometiendo más errores al predecir la clase 1.

En la figura 24 se presenta la curva ROC del modelo Self Training, se observa que el modelo tiene un buen rendimiento predictivo pues la curva se acerca mucho al borde superior izquierdo.

Figura 24

Curva ROC algorimo Self Training



Nota: La curva ROC señala un buen rendimiento del modelo Self Training.

Nuevamente, se presentan las métricas calculadas para el modelo Self Training en la tabla 7.

Tabla 7

Métricas de evaluación para Self Training

Métrica	Valor
AUC	0.998
Accuracy	0.985
Precisión	0.988
Recall	0.985
F1 Score	0.985

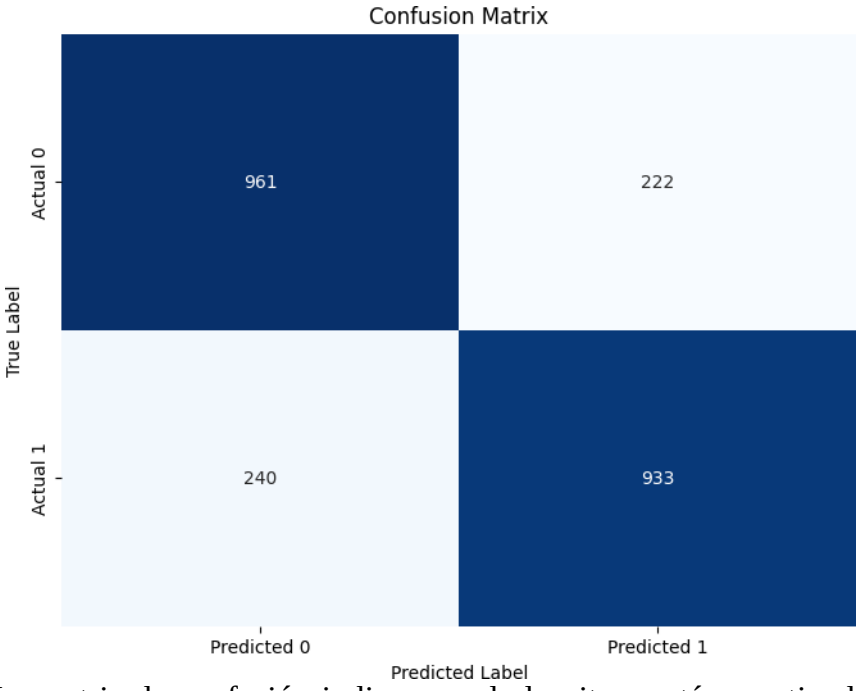
Nota: Esta tabla presenta los resultados del modelo Self Trainging

Label Propagation

En la figura 25 se presenta la matriz de confusión del modelo Label Propagation, se observa que el modelo se confunde al predecir ambas clases.

Figura 25

Matriz de confusión para algoritmo Label Propagation

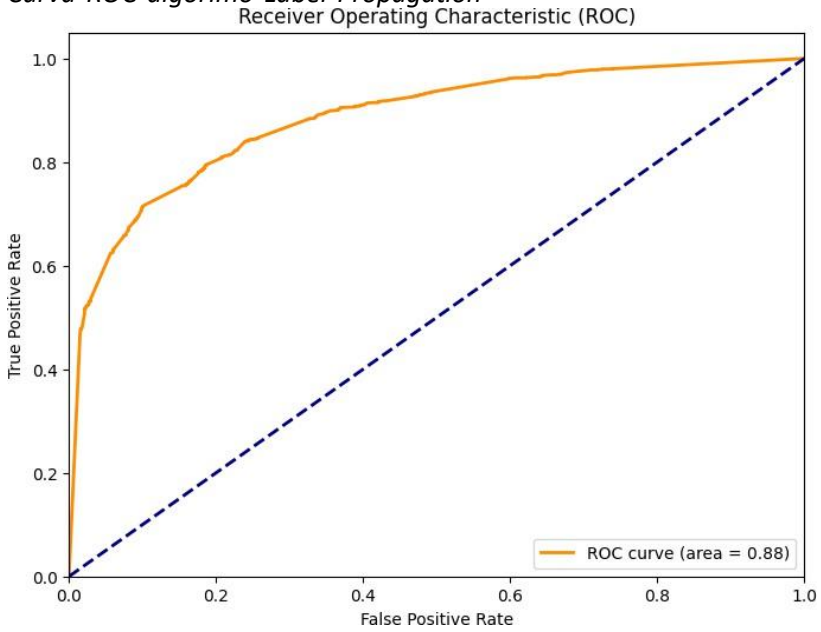


Nota: La matriz de confusión indica que el algoritmo está cometiendo errores al predecir ambas clases.

En la figura 26 se presenta la curva ROC del modelo Label Propagation, se observa que el modelo si bien no es malo, tampoco supera a otros modelos testeados.

Figura 26

Curva ROC algorimo Label Propagation



Nota: La curva ROC señala un rendimiento promedio del modelo Label Propagation.

Nuevamente, se presentan las métricas calculadas para el modelo Label Propagation en la tabla 8.

Tabla 8

Métricas de evaluación para Label Propagation

Métrica	Valor
AUC	0.884
Accuracy	0.803
Precisión	0.807
Recall	0.801
F1 Score	0.801

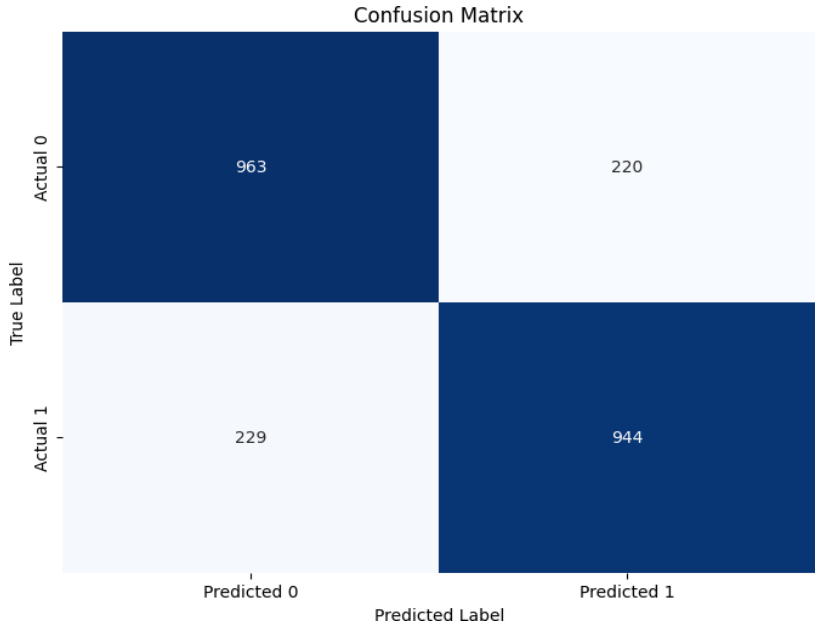
Nota: Esta tabla presenta los resultados del modelo Label Propagation

Label Spreading

En la figura 27 se presenta la matriz de confusión del modelo Label Spreading, y se observa un comportamiento parecido al modelo Label Propagation, con un poder predictivo regular.

Figura 27

Matriz de confusión para algoritmo Label Spreading



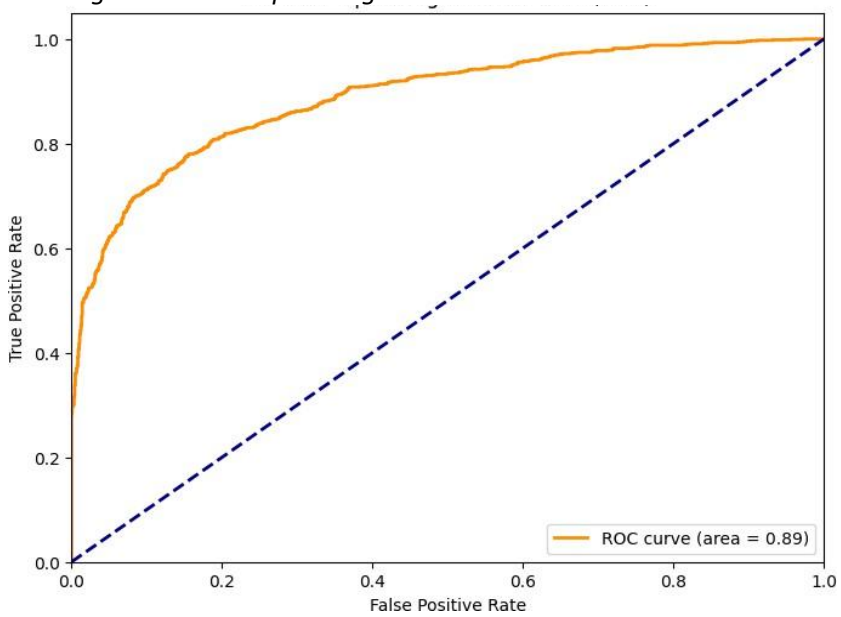
Nota: La matriz de confusión indica que el algoritmo está cometiendo errores al predecir ambas clases.

En la figura 28 se presenta la curva ROC del modelo Label Spreading, y nuevamente se observa que, si bien el modelo de no es malo, tampoco tiene un buen poder predictivo.

Nota: La curva ROC señala un rendimiento promedio del modelo Label Spreading.

Figura 28

Curva ROC algorimo Label Spreading



Nuevamente, se presentan las métricas calculadas para el modelo Label Spreading en la tabla 9.

Tabla 9

Métricas de evaluación para Label Spreading.

Métrica	Valor
AUC	0.888
Accuracy	0.809
Precisión	0.810
Recall	0.807
F1 Score	0.807

Nota: Esta tabla presenta los resultados del modelo Label Spreading

K-Means

Para el modelo entrenado mediante K-Means se utilizó clusterización. Como se puede observar en la Figura 29, cada punto representa una observación asignada a un clúster, y los colores distinguen los diferentes grupos. Los marcadores centrales indican las posiciones de los centroides calculados, destacando el punto medio de cada clúster. Este análisis visual permite identificar la distribución de los datos y la separación entre clústeres.

Tabla 10

Resultados del modelo K-Means

Métrica	Valor
Inercia	156839.1271
Silhouette Score	0.2262

Nota: Esta tabla presenta los resultados del modelo Label Spreading.

Interpretación:

El algoritmo K-means particionó los datos en 6 clusters, tal como se observa en la tabla 10. El Silhouette Score obtenido sugiere que los clusters tienen una separación moderada, pero la cohesión interna no es óptima.

Un Silhouette Score cercano a 0 indica que algunos puntos están equidistantes entre su propio cluster y el cluster vecino, lo cual refleja solapamiento entre clusters. La inercia es alta (156839.127), lo que indica que las distancias dentro de cada cluster aún son significativas, lo que sugiere que los datos tienen una estructura compleja y que K-means no logra agrupar completamente de manera eficiente.

DBSCAN

Para el modelo entrenado mediante DBSCAN, se encontró que el número de clusters es 12, pero 530 puntos fueron etiquetados como ruido (outliers), lo cual es común en algoritmos basados en densidad, especialmente cuando los datos tienen regiones dispersas. Como se puede observar en la Figura 30, la distribución de los puntos muestra una separación entre los clusters y los puntos de ruido.

Tabla 11

Resultados del modelo DBSCAN

Métrica	Valor
Número de clusters	12
Puntos de ruido	530
Silhouette Score (sin ruido)	-0.0769

Nota: Esta tabla presenta los resultados del modelo DBSCAN.

Interpretación:

El algoritmo DBSCAN encontró 12 clusters, pero 530 puntos fueron etiquetados como ruido (outliers), lo cual es típico en un modelo basado en densidad cuando los datos tienen regiones dispersas.

El Silhouette Score negativo (-0.0769) indica que los clusters encontrados no tienen cohesión ni separación clara, y muchos puntos probablemente están asignados incorrectamente a sus respectivos clusters. Esto sugiere que la distribución de densidad de los datos no es adecuada para que DBSCAN genere clusters bien definidos.

A pesar de ser robusto a ruido, la cantidad de ruido identificado y el bajo rendimiento (Silhouette Score negativo) reflejan que DBSCAN no fue efectivo para esta estructura de datos.

Autoencoders (Detección de Anomalías)

Para el modelo entrenado mediante Autoencoders, se detectaron 589 anomalías, lo cual indica que existen puntos en los datos que se consideran fuera de la distribución esperada.

Tabla 12

Resultados del modelo Autoencoders

Métrica	Valor
Número de anomalías detectadas	589
Silhouette Score	0.2031

Nota: Esta tabla presenta los resultados del modelo Autoencoders.

Interpretación:

El modelo de Autoencoders detectó 589 anomalías, lo cual indica que existen puntos en los datos que se consideran fuera de la distribución esperada.

El Silhouette Score obtenido (**0.2031**) es bajo, pero aún ligeramente mejor que el de DBSCAN, lo que sugiere que la representación aprendida por el autoencoder logra capturar algo de la estructura subyacente, aunque no lo suficiente para generar clusters bien definidos.

Al enfocarse en detectar anomalías y no en formar clusters tradicionales, los resultados son adecuados para la tarea específica de detección de outliers, aunque el score refleja solapamiento entre puntos no anómalos y anómalos.

Comparación y selección del mejor modelo para Aprendizaje Supervisado y Semi-Supervisado

Se comparan las métricas de validación de todos los algotimos de aprendizaje supervisado y semi - supervisado para escoger el modelo que tenga el mejor rendimiento.

Tabla 13

Comparación de métricas de evaluación para los algorimtos de aprendizaje supervisado y semi-supervisado.

Métrica	RF	SVM	CNN	SelfTraining	LabelPropagation	LabelSpreading
AUC	1.00	0.82	1.00	0.998	0.884	0.888
Accuracy	0.96	0.77	0.99	0.985	0.803	0.809
Precisión	0.94	0.79	0.99	0.988	0.807	0.810
Recall	0.94	0.64	0.99	0.985	0.801	0.807
F1 Score	0.96	0.71	0.99	0.985	0.801	0.807

Nota: Esta tabla presenta un resumen de los resultados de todos los modelos supervisado y semisupervisados.

Se puede validar en la tabla 13 que el modelo que mayor poder predictivo tiene es el algoritmo de redes neuronales convolucionales, ya que es el que optimiza todas las métricas

de evaluación.

Comparación y selección del mejor modelo para Aprendizaje No Supervisado

Los **Silhouette Scores** comparados son los siguientes:

Tabla 14

Comparación de los Silhouette Scores entre los modelos

Modelo	Silhouette Score
K-means	0.2262
DBSCAN	-0.0769
Autoencoders	0.2031

Nota: Esta tabla presenta un resumen de los resultados de los modelos no supervisados.

El modelo **K-means** obtuvo el mejor desempeño con un **Silhouette Score** de **0.2262**, lo que lo posiciona como el mejor método de clustering en este análisis.

El modelo **DBSCAN** presentó un rendimiento pobre con un **Silhouette Score** negativo (-0.0769), lo que refleja una falta de separación clara entre los clusters y una alta cantidad de ruido, lo que indica que DBSCAN no fue efectivo para esta estructura de datos.

Por último, los Autoencoders, diseñados para la detección de anomalías, tuvieron un Silhouette Score de 0.2031, lo que es ligeramente mejor que el de DBSCAN, pero inferior al de K-means. A pesar de esto, es un modelo adecuado para la tarea específica de detección de outliers.

Figura 29

Clusters para K-means

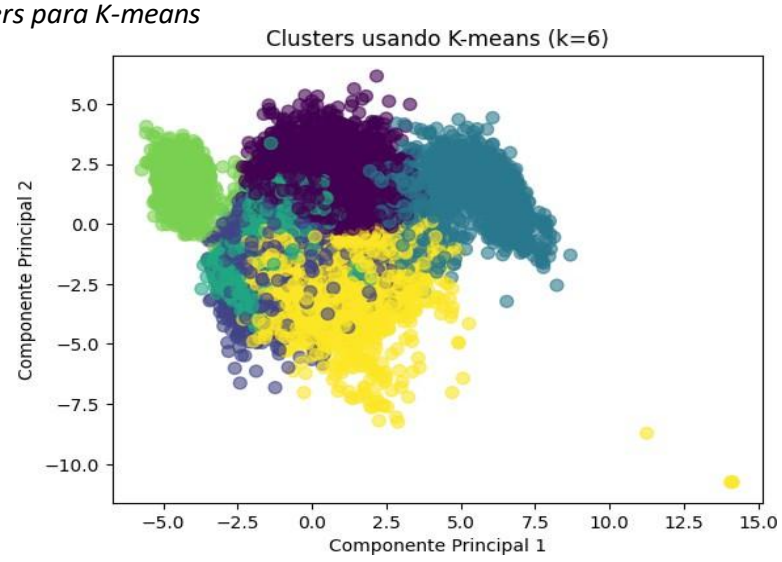
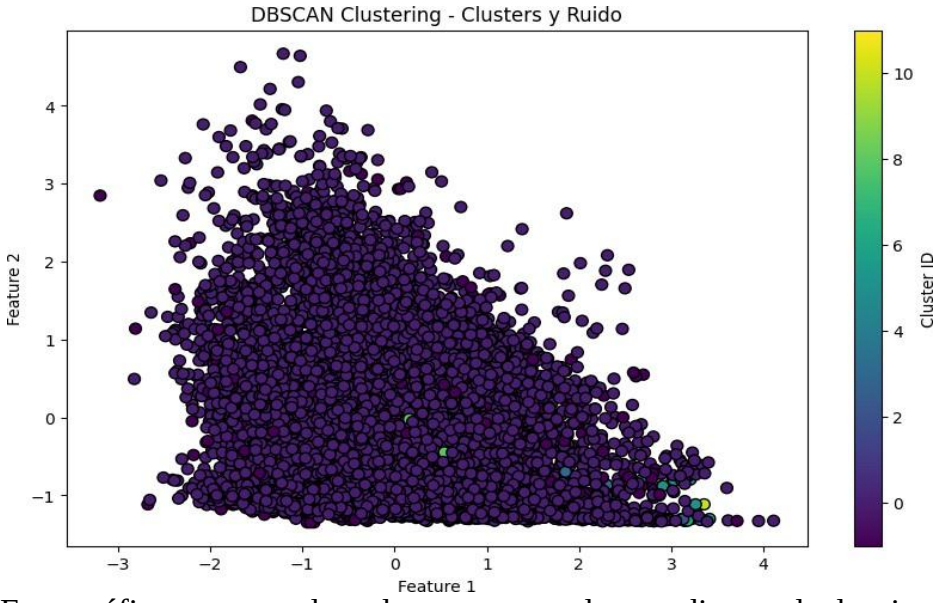


Figura 30

Clusters y ruidos generados mediante DBSCAN



Nota: Este gráfico muestra los clusters generados mediante el algoritmo DBSCAN, junto con los puntos etiquetados como ruido.

Capítulo 4

Conclusiones

Al finalizar este proyecto para detección de audios de DeepFake mediante algoritmos de aprendizaje supervisado, semi-supervisado y no supervisado, se obtuvieron las siguientes conclusiones:

Con respecto al modelo de Random Forest, se obtuvo un rendimiento notable para la detección de DeepFakes en audios. Se alcanzó un 94% de precisión en las predicciones, lo cual indica que el modelo está detectando adecuadamente los audios y los clasifica entre reales y falso con una exactitud casi perfecta; además, no se genera sobreajuste en el modelo. Se puede concluir que este modelo es un candidato ideal para implementar en la detección de audios falsos, debido a su gran rendimiento y efectividad.

Por otro lado, el modelo de Máquinas de Soporte Vectorial presentó un rendimiento regular, alcanzando un 79% de precisión en las predicciones de audios falsos. Este valor se logró gracias al ajuste de parámetros para obtener las mejores métricas, los cuales fueron `C=100` y `gamma=1` con una función kernel tipo `RBF`. Sin embargo, tiene deficiencias al momento de la clasificación en los audios, debido al 21% de error generado en sus predicciones. Se concluye que el modelo no es apto para implementación en la detección de audios falsos.

Se usaron técnicas de redes neuronales convolucionales para la detección de deepfakes en audios, siendo este algoritmo el que mejor resultados otorgó en la clasificación. Los resultados que se obtuvieron muestran que la implementación de una red neuronal convolucional unidimensional resulta adecuada para la clasificación de audios debido a su capacidad de procesar datos con estructuras secuenciales. El modelo CNN-1D logra aprovechar las características espectrales de los audios, como los coeficientes MFCC y el Chroma, lo que permitió que el modelo capture patrones significativos en los datos.

Además, la CNN muestra un comportamiento de pérdida estable a través de las épocas, lo que sugiere que el modelo no presenta problemas de sobreajuste, y es capaz de generalizar bien en datos que no ha visto.

Se exploraron técnicas de Aprendizaje Semi-Supervisado para detectar deepfakes en señales de audio. Para llevar a cabo estos experimentos, fue necesario simular un escenario real en el que se disponía de una cantidad limitada de datos etiquetados y un conjunto mayor de datos sin etiquetar. Esta simulación permitió evaluar la efectividad de los algoritmos de autoentrenamiento, propagación de etiquetas y propagación de etiquetas con suavizado en un entorno controlado.

Los resultados demuestran que estas técnicas son prometedoras para la detección de deepfakes, especialmente el algoritmo de self-learning que alcanzó una precisión de 0.985. Esto indica que, a través de un proceso iterativo de refinamiento del modelo utilizando datos pseudoetiquetados, es posible mejorar significativamente la capacidad de distinguir entre audio auténtico y sintético.

También se compararon tres enfoques de Aprendizaje No Supervisado para identificar deep fakes en audios, utilizando los modelos K-means, DBSCAN y Autoencoders. Los resultados se analizaron en función de la métrica del **Silhouette Score**, con el objetivo de alcanzar un valor mínimo de 0.5 en un plazo de tres meses.

K-means resultó ser el modelo más adecuado para la tarea de clustering, con un **Silhouette Score** de 0.2262, lo que indica un balance razonable entre la cohesión y separación de los clusters. Aunque la segmentación no fue perfecta debido a la complejidad de los datos, K-means logró proporcionar una agrupación significativa en comparación con otros modelos.

DBSCAN no mostró un rendimiento positivo, obteniendo un **Silhouette Score** negativo (-0.0769). Además, identificó una gran cantidad de puntos como ruido, lo que indica que el modelo no logró identificar clusters relevantes ni manejar adecuadamente la dispersión de los datos.

Autoencoders demostraron un desempeño moderado en la detección de anomalías, con un **Silhouette Score** de 0.2031. Aunque este modelo es más adecuado para la detección de outliers, no fue ideal para el clustering de datos en este caso, ya que no cumplió con los requisitos de segmentación para esta tarea específica.

Analizando todas las gráficas, métricas,desempeño de los algoritmos y aprendizaje supervisado/semi-supervisado, resulta conveniente escoger al modelo de redes neuronales convolucionales como método óptimo para la detección de audios de DeepFake, ya que presenta resultados más que satisfactorios con respecto a su rendimiento, poder de predicción y métricas de evaluación. Considerando aprendizaje no supervisado, la mejor opción para la detección de audios de DeepFake es el modelo K-Means, el cual presentó el mejor desempeño con respecto a clustering.

Recomendaciones

Preprocesamiento de datos y ajuste de hiperparámetros en Random Forest: se recomienda realizar preprocesamiento de los datos antes de implementar el algoritmo para evitar sobreajuste del modelo. Además, es buena práctica siempre definir la profundidad de los árboles para evitar demoras de procesamiento y consumo de recursos excesivo. De igual forma, esto ayuda a la visualización de los árboles de decisión generados.

Ajuste de hiperparámetros en SVM: se recomienda utilizar métodos para selección de parámetros, con el objetivo de ajustar de mejor manera el modelo para la predicción o clasificación de los datos. De esta manera, se puede mejorar tanto las métricas como el rendimiento del algoritmo.

Ajuste de hiperparámetros en las CNN: La arquitectura que se usó dio buenos resultados en la detección de deepfakes de audio. Sin embargo, se sugiere explorar mejoras adicionales, como ajustes en los hiperparámetros (tamaño de los kernels, número de filtros y profundidad de las capas) y el uso de técnicas avanzadas de interpretabilidad, como Grad-CAM o SHAP, para validar las decisiones del modelo.

Ajuste de hiperparámetros en las CNN: Es recomendable expandir el análisis que se hizo en este trabajo usando conjuntos de datos diversos para evaluar la robustez del modelo en diferentes escenarios. Asimismo, la integración de técnicas de aumento de datos y experimentos con arquitecturas híbridas podría contribuir a mejorar el desempeño en tareas más complejas de análisis de audio.

Exploración de métodos para simular datos: Considerando los

resultados obtenidos en este estudio, se recomienda enfocar futuras investigaciones en la exploración de estrategias más sofisticadas para generar entornos simulados de datos etiquetados y no etiquetados. Esto implica desarrollar técnicas de generación de datos sintéticos que sean cada vez más realistas y representativas de los datos del mundo real.

Exploración de técnicas de preprocesamiento de datos: Es fundamental profundizar en la investigación de métodos de limpieza y preprocesamiento de datos más robustos, con el objetivo de garantizar la calidad y la consistencia de los datos utilizados para entrenar los modelos. Por último, se sugiere explorar arquitecturas de aprendizaje profundo más complejas y adaptadas a las características específicas de los datos de audio, así como implementar procesos iterativos de entrenamiento y evaluación que permitan refinar continuamente los modelos y adaptarse a la evolución de las técnicas de generación de deepfakes.

Ajustar los parámetros de DBSCAN: Dado el rendimiento insuficiente de DBSCAN, es importante seguir ajustando los parámetros `eps` y `minPts` para mejorar la detección de clusters más pequeños o densos y reducir la cantidad de ruido, lo cual podría permitir obtener una segmentación más adecuada.

Evaluar otros métodos de clustering: Si el rendimiento de K-means sigue siendo subóptimo en algunos casos, se recomienda explorar otros métodos de clustering, como Agglomerative Clustering o Gaussian Mixture Models (GMM), que podrían ofrecer mejores resultados según la naturaleza de los datos.

Considerar la interpretabilidad de los resultados: Más allá de las métricas numéricas como el **Silhouette Score**, es importante priorizar modelos que ofrezcan una interpretación clara de los resultados. En aplicaciones prácticas, la comprensión de los grupos y la capacidad para visualizar los resultados son cruciales.

Bibliografía

Almutairi, Z. a. (2022). A review of modern audio deepfake detection methods: challenges and future directions. *Algorithms, 15*, pág. 155.

Bird, J. J. (2023). Real-time Detection of AI-Generated Speech for DeepFake Voice Conversion. *arXiv*.

Breiman, L. (October de 2021). Random Forests. *Machine Learning, 45*, págs. 5-32.

Capistrán, J. B. (2020). Deepfake: la imagen en tiempos de la posverdad. *Revista panamericana de comunicación, 2*, págs. 51-61.

Chapelle, O. a. (2006). Semi-supervised learning. *Cambridge, Massachusettes: The MIT Press View Article, 2*, págs. 1-3.

Chapelle, O. a. (2006). Semi-supervised learning. 2006. *Cambridge, Massachusettes: The MIT Press View Article, 2*, págs. 1-3.

Choudhury, S. a. (2022). Guess what moves: Unsupervised video and image segmentation by anticipating motion. *arXiv* .

González Carrizo, P. (2017). Estudio de sistemas de estimación automática de acordes en música digitalizada.

He, J. a. (2018). ReLU deep neural networks and linear finite elements. *arXiv*.

Heidari, A. a. (2024). A novel blockchain-based deepfake detection method using federated and deep learning models. *Cognitive Computation*, págs. 1-19.

Ige, A. O. (2022). A survey on unsupervised learning for wearable sensor-based activity recognition. *Applied Soft Computing, 127*, 109-363.

Jafari, M. a. (2020). Unsupervised learning and multipartite network models: a promising

approach for understanding traditional medicine. *Frontiers in Pharmacology, 11*, 13-19.

Jakkula, V. (2066). Tutorial on support vector machine (svm). *School of EECS, Washington State University, 37*, pág. 3.

James, G. a. (2023). Unsupervised learning. En *An introduction to statistical learning: with applications in Python* (págs. 503-556). Springer.

Johnson, D. G. (2021). What to do about deepfakes. *Communications of the ACM, 64*, págs. 33--35.

Larranaga, P. a. (1997). Tema 8. redes neuronales. *Redes Neuronales, U. del P. Vasco, 12*, pág. 17.

Lee, W. S. (2003). Learning with positive and unlabeled examples using weighted logistic regression. En *ICML* (Vol. 3, págs. 448-455).

Lyu, S. (2020). Deepfake detection: Current challenges and next steps. En IEEE, *2020 IEEE international conference on multimedia \& expo workshops (ICMEW)* (págs. 1-6).

McFee, B. a. (2015). librosa: Audio and music signal analysis in python. En *SciPy* (págs. 18-24).

Naeem, S. a. (2023). An unsupervised machine learning algorithms: Comprehensive review. *International Journal of Computing and Digital Systems*.

NewTechDojo. (2017). *Learn Support Vector Machine using Excel – Machine Learning Algorithm.*

Pinto, S. J. (2023). detection, Review of cybersecurity analysis in smart distribution systems and future directions for using unsupervised learning methods for cyber. *Energies, 16*, 16-51.

Rana, M. S. (2022). Deepfake detection: A systematic literature review. *IEEE access, 10*, págs. 25494--25513.

Rigatti, S. J. (January de 2017). Random Forest. *Journal of Insurance Medicine, 47*, págs. 31-39.

Rodríguez Copado, D. (2024). *Introducción al Machine learning, regresión y clasificación.* Universidad Internacional del Ecuador.

Rolf, B. a. (2024). A review on unsupervised learning algorithms and applications in supply chain management. *International Journal of Production Research*, 1-51.

Suarez, E. J. (2014). Tutorial sobre máquinas de vectores soporte (sVM). *Tutorial sobre Máquinas de Vectores Soporte (SVM), 1*, págs. 1-12.

Suwajanakorn, S. a.-S. (2017). Synthesizing obama: learning lip sync from audio. *ACM Transactions on Graphics (ToG), 36*, págs. 1-13.

Thomas, S. a. (2014). Analyzing convolutional neural networks for speech activity detection in mismatched acoustic conditions. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (págs. 2519--2523). IEEE.

Yamashita, R. a. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging, 9*, págs. 611-629.

Yang, A. Y. (2016). *Random Forest* (Vol. 15). Obtenido de https://math.mcgill.ca/yyang/resources/doc/randomforest.pdf

Zhou, D. a. (2003). Learning with local and global consistency. *Advances in neural information processing systems, 16*.

Zhu, X. a. (2022). *Introduction to semi-supervised learning.* Springer Nature.

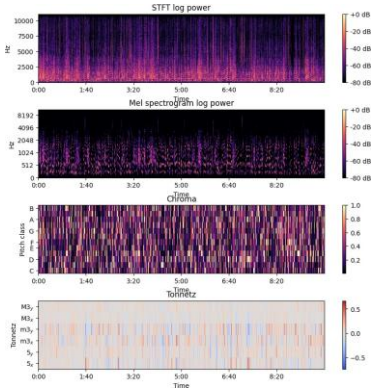
Anexos

Descomposición espectral de audios

En esta sección se muestran los gráficos de descomposición espectral de los audios analizados.

Figura 31

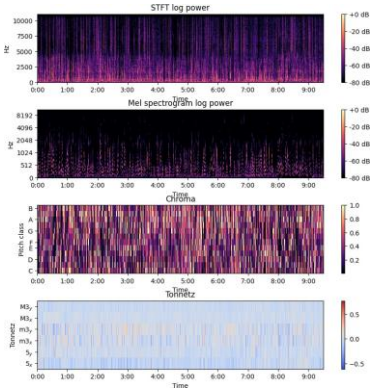
Descomposición espectral del audio de Joe Biden



Nota: En la figura se muestra la descomposición espectral del audio de Joe Biden.

Figura 32

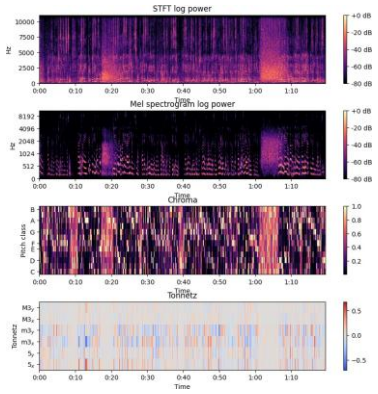
Descomposición espectral del audio de Linus Sebastian



Nota: En la figura se muestra la descomposición espectral del audio de Linus Sebastian.

Figura 33

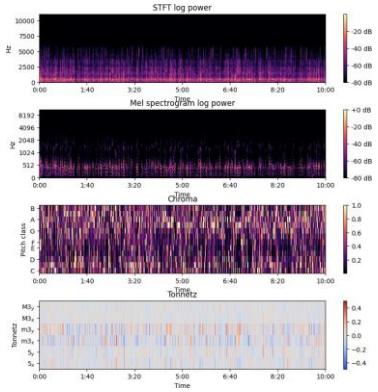
Descomposición espectral del audio de Margot Robbie



Nota: En la figura se muestra la descomposición espectral del audio de Margot Robbie.

Figura 34

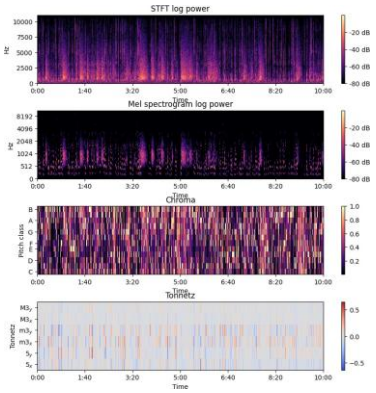
Descomposición espectral del audio de Elon Musk



Nota: En la figura se muestra la descomposición espectral del audio de Elon Musk.

Figura 35

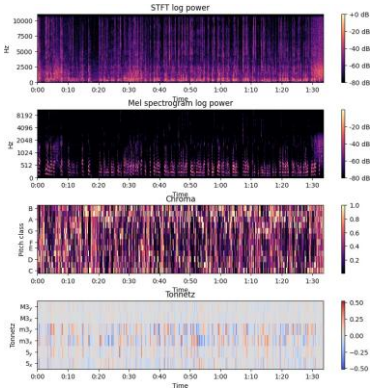
Descomposición espectral del audio de Barack Obama



Nota: En la figura se muestra la descomposición espectral del audio de Barack Obama.

Figura 36

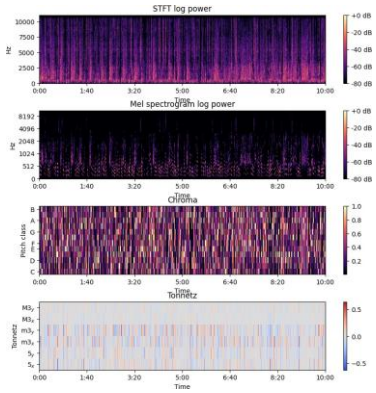
Descomposición espectral del audio de Ryan Gosling



Nota: En la figura se muestra la descomposición espectral del audio de Ryan Gosling.

Figura 37

Descomposición espectral del audio de Donald Trump



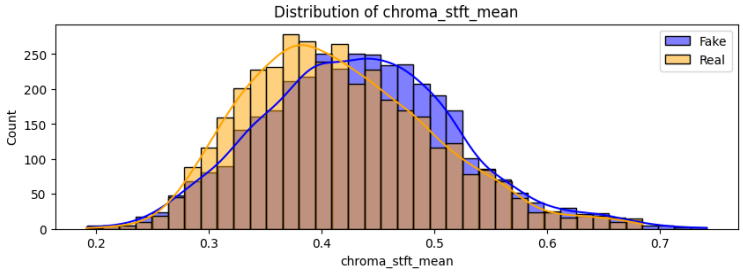
Nota: En la figura se muestra la descomposición espectral del audio de Donald Trump.

Distribución de variables vs. variable objetivo

A continuación, se muestran las distribuciones de cada variable obtenida a partir de la descomposición espectral de los audios vs. la variable objetivo.

Figura 38

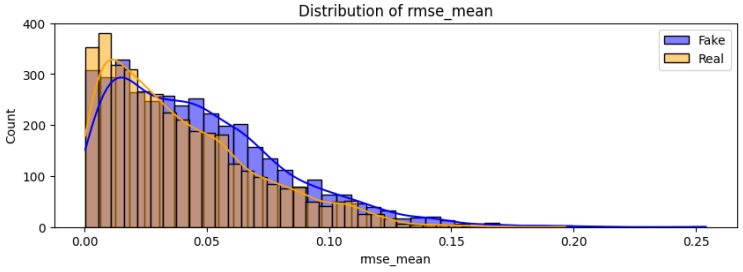
Distribución de variable Chroma



Nota: En la figura se muestra la distribución de la variable Chroma.

Figura 39

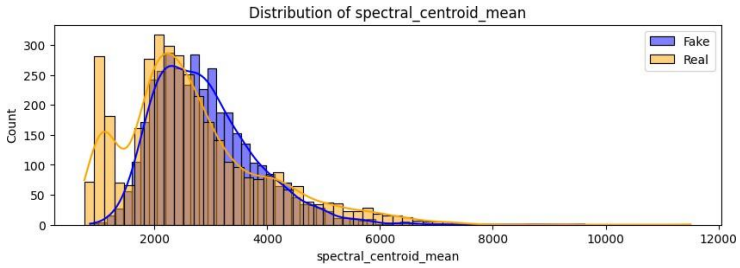
Distribución de variable RMSE



Nota: En la figura se muestra la distribución de la variable RMSE.

Figura 40

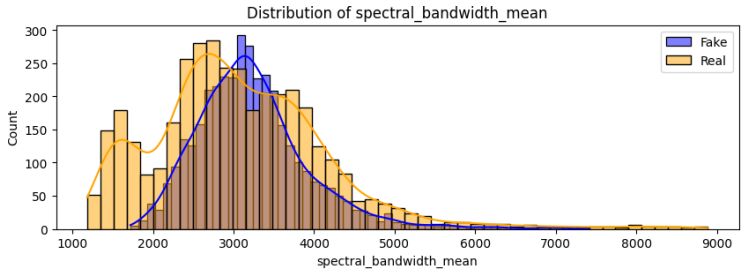
Distribución de variable Spectral Centroid



Nota: En la figura se muestra la distribución de la variable Spectral Centroid.

Figura 41

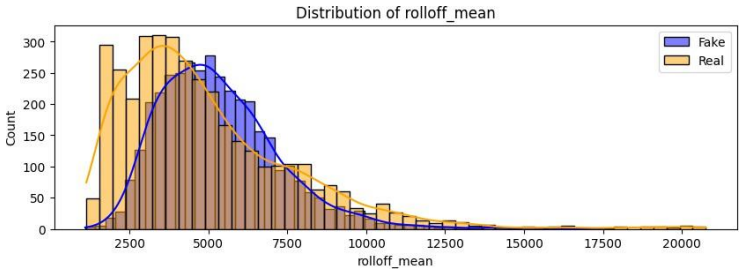
Distribución de variable Spectral Bandwidth



Nota: En la figura se muestra la distribución de la variable Spectral Bandwidth.

Figura 42

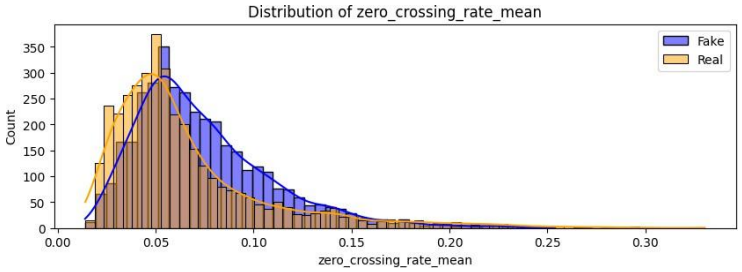
Distribución de variable Rolloff



Nota: En la figura se muestra la distribución de la variable Rolloff

Figura 43

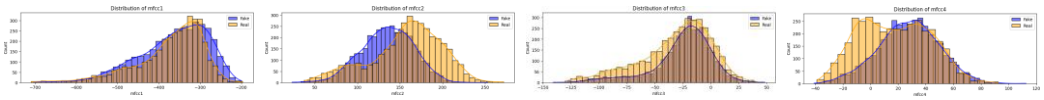
Distribución de Zero variable Crossing Rate



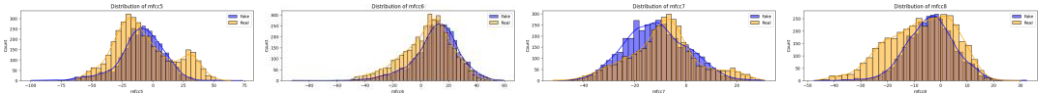
Nota: En la figura se muestra la distribución de la variable Zero Crossing Rate.

Figura 44

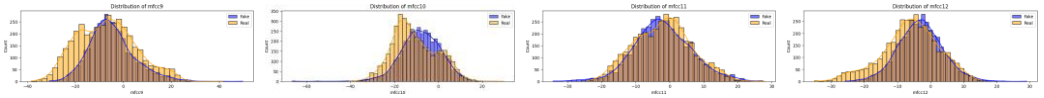
Distribución de variable MFCC1 - MFCC20



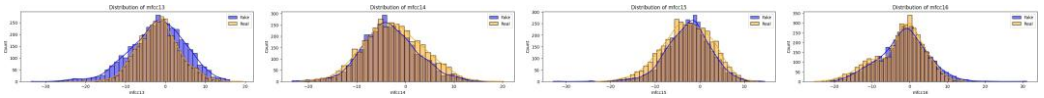
(a) *MFCC1* **(b)** *MFCC2* **(c)** *MFCC3* **(d)** *MFCC4*



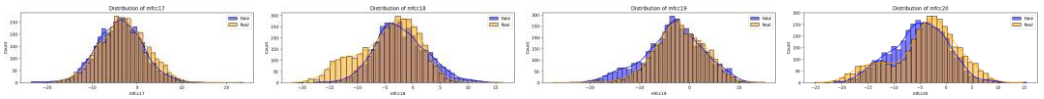
(e) *MFCC5* **(f)** *MFCC6* **(g)** *MFCC7* **(h)** *MFCC8*



(i) *MFCC9* **(j)** *MFCC10* **(k)** *MFCC11* **(l)** *MFCC12*



(m) *MFCC13* **(n)** *MFCC14* **(o)** *MFCC15* **(p)** *MFCC16*



(q) *MFCC17* **(r)** *MFCC18* **(s)** *MFCC19* **(t)** *MFCC20*

Nota: En la figura se muestra la distribución de la variables creadas a partir de la descomposición espectral.