UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMATICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

TESIS:

SISTEMA PARA DETECCIÓN DE AVALANCHAS USANDO UN SOLO SENSOR INFRASÓNICO Y ALGORITMOS DE MACHINE LEARNING

PRESENTADO POR:

Br. Cristian Adriel Benites Condori

PARA OPTAR AL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

ASESOR:

Ing. Arizaca Cusicuna, Jorge Luis

FINANCIADO POR:

Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña (INAIGEM)

CUSCO - PERÚ

2023

INFORME DE ORIGINALIDAD

(Aprobado por Resolución Nro.CU-303-2020-UNSAAC)

El que suscribe, asesor del trabajo de investigación titulado "**SISTEMA PARA DETECCIÓN DE AVALANCHAS USANDO UN SOLO SENSOR INFRASÓNICO Y ALGORITMOS DE MACHINE LEARNING**", presentado por el bachiller CRISTIAN ADRIEL BENITES CONDORI, con DNI número 73049941 para optar al Titulo Profesional de Ingeniero Electrónico.
Informo que el trabajo de investigación ha sido sometido a revisión por tres veces, mediante el software antiplagio, conforme al Artículo 6° del presente reglamento y de la evaluación de originalidad se tiene un porcentaje de **8%**.

Evaluación y acciones del reporte de coincidencia para trabajos de investigación, tesis, textos, libros, revistas, artículos científicos, material de enseñanza y otros (Art. 7, inc 2 y 3)

Porcentaje	Evaluación y acciones.	Marque con X
Del 1 al 10 %	No se considera plagio.	X
Del 11 al 30%	Devolver al usuario para las correcciones.	
Mayores a 31 %	El responsable de la revisión del documento emite un informe al inmediato jerárquico, quien a su vez eleva el informe a la autoridad académica para que tome las acciones correspondientes. Sin perjuicio de las sanciones administrativas que correspondan de acuerdo a ley.	

Por tanto, en mi condición de Asesor, firmo el presente informe en señal de conformidad y adjunto la primera hoja del reporte del Sistema Antiplagio.

Cusco, 24 de agosto del 2023

Jorge Luis Arizaca Cusicuna
DNI: 42348906
https://orcid.org/0000-0003-2658-5482

Se adjunta:
1. Reporte generado por el Sistema Antiplagio.
2. Enlace del Reporte Generado por el Sistema Antiplagio:
https://unsaac.turnitin.com/viewer/submissions/oid:27259:257670445?locale=es-MX

NOMBRE DEL TRABAJO
Tesis_infrasound_Cristian_Benites__Version_23276 Final.pdf

AUTOR
Cristian Benites

RECUENTO DE PALABRAS
46349 Words

RECUENTO DE CARACTERES
238746 Characters

RECUENTO DE PÁGINAS
191 Pages

TAMAÑO DEL ARCHIVO
13.3MB

FECHA DE ENTREGA
Aug 24, 2023 9:41 AM GMT-5

FECHA DEL INFORME
Aug 24, 2023 9:44 AM GMT-5

8% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base d

- 6% Base de datos de Internet
- 1% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossr
- 5% Base de datos de trabajos entregados

Excluir del Reporte de Similitud

- Material bibliográfico
- Coincidencia baja (menos de 8 palabras)

Dedicatoria

Esta tesis esta dedicada a mis padres Libio y Genara, quienes me proporcionaron el apoyo necesario desde el inicio hasta la culminación de este trabajo, y quienes fueron mi inspiración para seguirme superando.

A mis dos hermanos, Libio Antero y Rosaliz, por su colaboración y por sus palabras de aliento constante.

Al Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña INAIGEM por hacer posible la realización de esta tesis, a mi coasesor, el Ing. Robert Alvarado Lugo, y al subdirector de la dirección de información y análisis, el Dr. Christian Yarlequé Galvez, los cuales me apoyaron con su experiencia y me brindaron su asesoramiento en la elaboración e implementación del presente trabajo; y en general, un agradecimiento a todas las personas que me ayudaron y acogieron durante el desarrollo de la tesis en la ciudad de Huaraz.

Resumen

La detección automática de las avalanchas es una herramienta muy útil para poder reducir los riesgos que generan estos desastres naturales. Aunque en la literatura existen distintos métodos para la detección de avalanchas, su detección mediante infrasonido tiene algunas ventajas frente a otro métodos, debido sobretodo al bajo nivel de atenuación de las ondas de sonido de baja frecuencia al viajar a través de distintos materiales, lo que hace que pueda ser detectado a varios kilómetros del lugar de donde sucedió la avalancha. La gran mayoría de sistemas de detección por infrasonido en la literatura utilizan arreglos de sensores, estos son caros y difíciles de implementar por lo que en este trabajo se hizo uso de un solo sensor, las limitaciones que puede presentar un sistema compuesto de un solo sensor fueron compensadas utilizando algoritmos de *Machine Learning* para las tareas de clasificación. Para la implementación del sistema se recolectó data de infrasonido de más de 100 eventos de avalanchas en la laguna Palcacocha, se realizó la búsqueda del patrón de características y el algoritmo de entrenamiento para lograr la obtención del modelo de *Machine Learning* óptimo, se hicieron pruebas entrenando modelos bajo distintos algoritmos llegando a la conclusión de que el algoritmo óptimo es el de *Support Vector Machine*, logrando una efectividad teórica del 92.27 %. Luego, se implementaron, el procesamiento en tiempo real, el almacenamiento de la data en una base de datos y la visualización de la data a fin de monitorear los resultados de forma remota. Finalmente se hicieron pruebas piloto en la laguna Palcacocha, obteniendo un porcentaje de efectividad en la detección de avalanchas del 86.47 %. El sistema desarrollado es definitivamente más barato y fácil de implementar que los sistemas que utilizan arreglos de sensores, además de que se obtienen mejores resultados en la detección a diferencia de otros que no utilizan *Machine Learning*.

Palabras Clave: Cordillera Blanca, Raspberry Pi, Avalanchas, Infrasonido, Detección automática, Detección en tiempo real, Machine learning.

Abstract

The automatic avalanche detection is a useful tool for decreasing the risk that is generated from this natural disaster. Although differents methods for avalanche detection can be found in the literature, infrasound detection has some advantages over other methods, mostly because the low level of attenuation that the low frequency sound waves has while traveling through diverse materials, this allows the remote detection to several kilometers from where the avalanche was produced. The vast majority of infrasound detection systems in the literature use sensor arrays, these are expensive and difficult to implement so in this work just one sensor was used, the limitations of using a single sensor was compensated with the use of Machine Learning algorithms for the classification task. For the system implementation, data from more than 100 avalanche ocurrences in the Palcacocha Lake was collected, the feature pattern and the training algorithm to achieve the most optimal Machine Learning model were searched, tests were made training models with differents algorithms, coming to the conclusion that the most optimal algorithm is Support Vector Machine, thus, achieving a theoretical accuracy of 92.27 %. Then, real time processing, the data storage in a database and the data visualization were implemented in order to monitor the results remotely. Finally, pilot tests were made in the Palcacocha Lake, achieving an accuracy of 86.47 %. The system developed in this work was definitely cheaper and easier to implement than systems that use array sensors and, moreover, better results were found in the detection than other systems that don't use Machine Learning.

Keywords: Cordillera Blanca, Raspberry Pi, Avalanches, Infrasound, Automated detection, Real-time detection, Machine learning.

Introducción

Las avalanchas de nieve amenazan la seguridad de personas e infraestructura, especialmente en el Perú, donde ya han sucedido distintos desastres relacionados a aludes y avalanchas. Diferentes estrategias se han desarrollado en la literatura y el mercado para poder reducir los riesgos que representan las avalanchas, estas incluyen monitoreo visual y detección temprana. El monitoreo visual, aunque es relativamente fácil de implementar no es una solución definitiva, ya que se pierde visibilidad en distintas situaciones como durante niebla, lluvia o en la noche, por otro lado, la detección automática es una solución más práctica ya que no necesariamente depende de la visibilidad de la avalancha, sino de otras señales como pueden ser el sonido o las vibraciones sísmicas producidas por las avalanchas. En el Perú, particularmente, existen muy pocos sistema de monitoreo relacionados al tema de avalanchas, en este sentido, se ha planteado el presente trabajo, a fin de abordar esta problemática, además de explorar un tema muy poco visto en el Perú.

La presente propuesta de investigación consiste en la implementación de un sistema de detección de avalanchas en tiempo real en la laguna Palcacocha, este sistema será capaz de detectar avalanchas aprovechando las señales de infrasonido que se generan cuando se produce un desprendimiento de masa glaciar, dicha detección se realizará mediante algoritmos de *Machine Learning* a fin de optimizar las tareas de clasificación.

Índice

Índice de figuras

Índice de tablas

Índice de códigos

Glosario y Acrónimos

INAIGEM: Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña

IDA: Infrasound Detection of Avalanches

FFT: Fast Fourier Transform

ADC: Analog-to-Digital Converter

Accuracy: Métrica para evaluar modelos de clasificación de Machine Learning

SVM: Supervised Machine Learning

KNN: k-Nearest Neighbors

SEED: Standard for the Exchange of Earthquake Data

URL: Uniform Resource Locator

SSH: Secure SHell

SFTP: SSH File Transfer Protocol

TCP/IP: Transmission Control Protocol/Internet Protocol

DNS: Domain Name System

UTC: Coordinated Universal Time

CSV: comma-separated values

TSFEL: Time Series Feature Extraction Library

JSON: JavaScript Object Notation

BER: Band Energy Ratio

CLI: Command-Line Interface

SQL: Structured Query Language

CMD: Abreviación de la palabra Command, es un intérprete de comandos del sistema operativo Windows

UDF: User Defined Function

I. Generalidades

1.1. Marco referencial

1.1.1. Titulo

"SISTEMA PARA DETECCIÓN DE AVALANCHAS USANDO UN SOLO SENSOR INFRASÓNICO Y ALGORITMOS DE MACHINE LEARNING"

1.1.2. Responsable

Cristian Adriel Benites Condori

1.1.3. Asesor

MSc. Jorge Luis Arizaca Cusicuna

1.1.4. Ámbito Geográfico

El desarrollo de la tesis tendrá lugar en la laguna Palcacocha, que es una laguna de origen glaciar de la Cordillera Blanca del Perú, esta se encuentra ubicada en el departamento de Ancash, provincia de Huaraz .

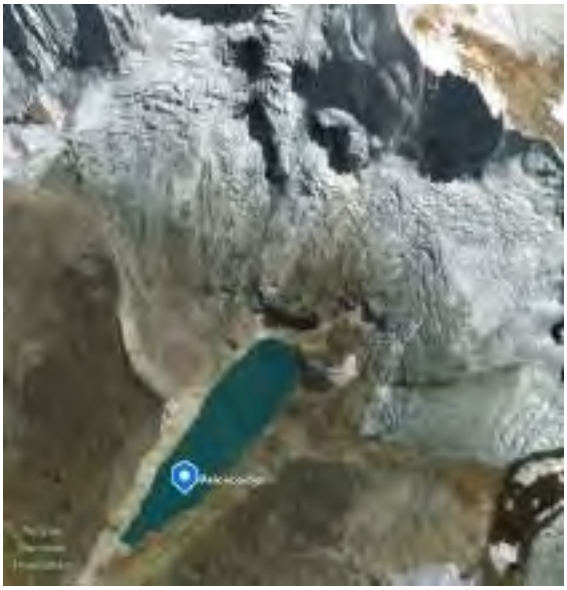

Figura 1: Escenario de desarrollo de la tesis.

1.2. Planteamiento del problema

1.2.1. Problemática

Las avalanchas de nieve son fenómenos naturales que pueden generar daños materiales e incluso pueden costar vidas humanas, especialmente en el Perú, donde tenemos antecedentes de tragedias como la ocurrida en la ciudad de Yungay el año 1970, la cuál trajo consigo desastre y tomo la vida de 50000 personas, este problema latente se agravará aún más con el paso del tiempo debido al calentamiento global y el deshielo de los glaciares; a todo esto se añade la explosión demográfica y de infraestructura que se ha dado en estos últimos años en los valles de alta montaña, que hacen que se incremente el peligro de una potencial pérdida de vidas humanas y desastres de gran alcance [1].

En 1941, se produjo la ruptura de un dique de la laguna Palcacocha, como consecuencia del desprendimiento de grandes masas de hielo de los Nevados de Pucaranra y Palcaraju; este desprendimiento ocasionó un desborde violento de la laguna, el flujo de agua combinado con las masas erosionadas de las laderas llegó a la ciudad de Huaraz, produciendo grandes pérdidas tanto materiales como humanas cobrándose la vida de alrededor de 1800 personas [2].

Figura 2: Consecuencias del aluvión en la ciudad de Huaraz (1941).

Fuente: [2]

Como el desastre de 1941, existen varios sucesos parecidos a lo largo de la historia de la ciudad Huaraz, y también podemos encontrar muchos ejemplos más a lo largo de todo el territorio peruano. Existen muchos factores por los cuales estos desastres fueron particularmente perjudiciales, pero uno de ellos es que nunca existió un sistema de alerta temprana que pudiera haber ayudado a prevenir, como mínimo, la perdida de vidas humanas, esto es algo que esta comenzando a cambiar en este nuevo siglo [2].

Actualmente existen muchos sistemas de detección de avalanchas implementados en diferentes lugares del mundo, incluso ya existen sistemas de detección comerciales, como el que ofrece la empresa suiza Wyssen, cuyo sistema está basado en un arreglo de sensores infrasónicos (IDA®), por supuesto esto con un costo económico bastante alto, que llega a varios miles de dolares dependiendo de la complejidad de la zona del despliegue.

En Huaraz-Perú, en la laguna Palcacocha existe un sistema de monitoreo implementado por el Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña (INAIGEM), este sistema consiste de una estación meteorológica, una cámara de videovigilancia y un sensor de nivel de agua en la laguna. El sensor de nivel sirve para monitorear el estado de la laguna y poder prevenir un evento de desborde de agua; los sensores de la estación meteorológica reporta data útil para el monitoreo de variables relacionados al clima en la zona como son: temperatura, precipitación, humedad, velocidad

de viento, etc. Estas dos herramientas son útiles para el propósito específico que fueron diseñados pero por si solas no pueden ser utilizadas en la detección de avalanchas, para ello se requeriría de un análisis más profundo de la data disponible, como podría ser la construcción de un modelo predictivo de avalanchas a partir de las diferentes variables obtenidas de los sensores.

Por otro lado, también se dispone de un sistema de videovigilancia, este sistema es capaz de monitorear en tiempo real los eventos que suceden en la laguna y los nevados, dentro de su campo de visión, esto incluye avalanchas y desprendimientos de masas hacia la laguna, aunque este sistema es muy útil, tiene las inconveniencias de que no es posible utilizarlo cuando se presentan obstáculos visuales de la naturaleza como puede ser: neblina o falta de visión durante la noche.

En resumen, aunque en el Perú siempre ha existido problemas relacionados a avalanchas y desprendimientos de masas, aún no se cuenta con un sistema de monitoreo de avalanchas que sea funcional durante las 24 horas del día, debido a que, entre otros factores, la sensibilidad de los instrumentos que actualmente se han implementado no es adecuada para la detección de las avalanchas, sobretodo en la noche.

1.2.2. Formulación del problema

¿De que manera podemos monitorear y detectar, durante las 24 horas del día en un entorno montañoso, los eventos de avalanchas y desprendimientos de masa sobre la laguna Palcacocha a fin de poder prevenirlos y disminuir los riesgos de desastres relacionados a estos?

1.2.3. Problema General

La falta de herramientas para la detección y monitoreo de eventos de avalancha en el Perú que además funcionen de forma automática y efectiva durante las 24 horas del día.

1.3. Justificación

Como ya se ha explicado en la problemática, las avalanchas son fenómenos naturales muy peligrosos que significan un riesgo de peligro para las poblaciones que habitan las partes medias y altas de los valles de alta montaña. Conociendo esto, los métodos de

prevención y detección temprana de tales desastres deben ser considerados una necesidad, esta investigación se ha planteado con la finalidad de abordar esta problemática.

La detección de avalanchas no es un tema nuevo, en diferentes países y a través de distintas investigaciones se han implementado distintas formas de detección de avalanchas: con sensores sísmicos [3], con radares [4], y con sensores infrasónicos [5], y también sistemas integrados comerciales [6].

Para este trabajo se decidió utilizar infrasonido, y la razón de porque se utilizó este método de detección es que presenta ciertas ventajas con respecto a otros métodos de detección según comparativas realizadas en otros trabajos [7] [8], principalmente el hecho de que la detección remota del infrasonido puede hacerse a varios kilómetros de distancia desde donde sucedió la avalancha (hasta 3Km [9]) a diferencia de un sensor sísmico que debe estar a una distancia cercana del lugar donde sucedió la avalancha (menor a 100 m); en cuanto a la detección por radar; aunque la efectividad de este método en cuanto a detección es superior y su distancia de detección es parecida a la del método por infrasonido su instalación es mucho mas compleja y costosa, además de que su rango de operación esta limitado a un sector definido mientras que los sensores de infrasonido pueden detectar señales en un rango de 360°.

Muchos de los trabajos que se encuentran en la literatura con respecto a detección de avalanchas por infrasonido que se realizaron hasta el momento se hicieron utilizando arreglos de sensores [10] [9] [11], por otro lado, sistemas que utilizan un solo sensor también existen en la literatura, por ejemplo en algunos estudios se han comparado la efectividad y las limitaciones de un sistema compuesto por un sensor en comparación de un arreglo de sensores [12] [13], entre las limitaciones del uso de un solo sensor, esta el hecho de que no se puede determinar la dirección de la avalancha, y también el bajo nivel de señal a ruido.

En este trabajo se plantea el uso de un solo sensor, tomando en cuenta las limitaciones que eso implica. Para poder cubrir dichas limitaciones, se utilizará una herramienta que se ha estado popularizando estos últimos años: La Inteligencia Artificial y el *Machine Learning*. Esta aproximación no es del todo nueva, existe un estudio donde se ha hecho uso de *Machine Learning* para la detección de avalanchas por infrasonido y se ha demostrado que utilizar esta tecnología puede disminuir significativamente los falsos positivos [14], aunque cabe mencionar que para dicho estudio se empleo un arreglo de sensores, a

diferencia de la solución con un solo sensor que se plantea en este trabajo de investigación.

1.4. Antecedentes Generales

La detección de avalanchas utilizando infrasonido no es un tema nuevo, existen distintas soluciones desarrolladas tanto por parte de investigadores como por parte del sector privado con fines comerciales, se desarrollaran a continuación algunas de estas soluciones:

1.4.1. Sistema de detección de avalanchas IDA®

Este es un sistema desarrollado por la empresa suiza Wyssen, permite monitorear la actividad de avalanchas dentro de un radio de 3 a 5 Km. Este sistema esta basado en un arreglo de sensores de infrasonido que consiste en 4 o 5 sensores que están desplegados en forma de estrella alrededor de una unidad central. Usa algoritmos y filtros para diferenciar avalanchas de otras fuentes de sonido.

Figura 3: Sistema de detección de avalanchas IDA® Infrasound Detection System.

1.4.2. Sistemas de monitoreo y alarma ante amenazas naturales Geopraevent

La empresa Geopraevent desarrolla e implementa sistemas de monitoreo y alarma ante varios tipos de amenazas naturales, de forma particular, también se encarga de sistemas de monitoreo y detección de avalanchas de nieve, para ello utilizan no solo sensores de

infrasonido, sino también de geófonos y radares, la información de estos sensores es procesada y puede ser monitoreada a través de un portal web.

Figura 4: Sistema de detección y monitoreo de avalanchas instalado por Geopraevent alrededor de la montaña Piz Chapisun en el valle de Lower Engadin (Suiza).

1.4.3. Publicaciones científicas

El infrasonido ha sido estudiado para la detección de avalanchas desde los años 90 [15] [16]. Aunque, debido a su bajo performance [17], nunca había sido un referente en la detección de avalanchas en comparación de otros métodos de detección, uno de los factores que ha mermado su desempeño desde prácticamente los inicios es el ruido inducido por el viento en las mediciones. El viento puede generar ruido en todas las frecuencias del espectro del infrasonido, además de que su naturaleza es aleatoria por lo que ha supuesto un gran reto a resolver, entre las soluciones propuestas para solventar este problema están: el uso de pantallas de viento (*Windscreens*) [18], arreglos de tubos (*Pipe Arrays*) [19], utilización de arreglos geométricos de varios sensores para incrementar el nivel de señal a ruido [11], siendo este último combinado con alguno de los otros, los métodos que más se utilizan.

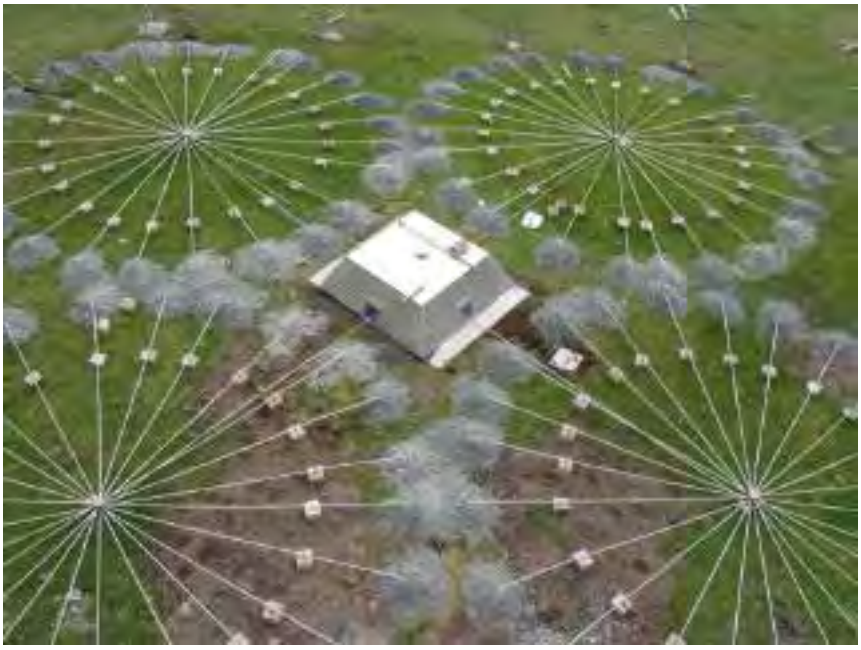

Figura 5: Arreglo de sensores utilizando un arreglo de tubos en cada sensor para disminuir el ruido producido por el viento.

Fuente: Estación de Monitoreo de Infrasonido IS49, Tristan da Cunha, Reino Unido

En lo que respecta a los métodos de detección en sí, es decir, los métodos con los que se pueden diferenciar las señales correspondientes a avalanchas de otras fuentes de infrasonido (como serían los sismos, meteoros, truenos), se suelen utilizar algoritmos. El algoritmo más utilizado en la detección por infrasonido está basado en umbrales [9] [17] [12], este consiste en umbrales definidos dentro de los cuales deben encontrarse ciertos criterios cuantificables de una señal para que esta pueda ser considerada como un evento de avalancha, por ejemplo: que el evento detectado debe durar mas de 10 s, o que la amplitud pico de todos los sensores debe ser mayor a 0.05 Pa [17], y demás criterios que varían entre los distintos estudios y trabajos. Estos algoritmos tienen el problema particular de que ocasionan un alto porcentaje de falsas alarmas, por ello ha surgido otras aproximaciones para solucionar este problema. En una investigación realizada el año 2014 en un instituto suizo [14] se ha encontrado que el porcentaje de falsas alarmas puede ser disminuido considerablemente si se utiliza algoritmos de inteligencia artificial y *Machine Learning*, la efectividad de este estudio fue del 57 % para la detección de eventos de avalancha y del 10 % de falsas alarmas, esto debido a que avalanchas de nivel menor son difíciles de detectar.

Las técnicas de detección basadas en infrasonido son muy prometedoras, y aunque se

hayan superado ya muchas limitantes en cuanto al desempeño de los sensores, la gran tarea se encuentra ahora en la parte del procesamiento de señales.

1.5. Objetivos

1.5.1. Objetivo general

Implementar un sistema de detección de avalanchas de origen glaciar utilizando un solo sensor de infrasonido en la laguna Palcacocha-Huaraz.

1.5.2. Objetivos específicos

- Implementar un sistema de monitoreo que permita el análisis en tiempo real y de forma remota de las señales de infrasonido de las zonas aledañas a la laguna Palcacocha.
- Organizar un set de datos con información del infrasonido de eventos de avalanchas reales, procurando la menor cantidad de ruido sobre la toma de datos.
- Determinar el algoritmo, los hiperparámetros y el vector de características que resulten en el modelo de *Machine Learning* más óptimo.
- Calcular la efectividad del sistema de detección a través de la realización de pruebas piloto.

1.6. Metodología de Investigación

1.6.1. Tipo de Investigación

La presente investigación se encuentra clasificada como una investigación de tipo aplicada, descriptiva y cuantitativa.
APLICADA, puesto que busca aportar en la solución a una problemática específica, que son las pocas herramientas que existen en el Perú para la prevención de desastres relacionados a avalanchas; DESCRIPTIVA, puesto que se busca especificar las características y patrones en los eventos de avalanchas reales de la data recolectada; y CUANTITATIVA, puesto que en esta investigación la recolección de datos se basa en instrumentos estandarizados,

además de que la naturaleza de los datos es cuantitativa (datos numéricos) y el análisis de estos datos es sistemático, estandarizado y posterior a su recolección.

1.6.2. Método

El método que se usara es el experimental; en este método se manipula el valor de una variable (variable independiente), para controlar el aumento o disminución de esta y observar su efecto en otra variable (variable dependiente).

1.6.3. Identificación de Variables y sus Indicadores

1.6.3.1. Variables

- **Variable independiente:** Modelo de *Machine Learning*
- **Variable dependiente:** Detección de eventos de avalancha.

1.6.3.2. Indicadores

- **Indicadores de la variable independiente:**

  Indicador: Cantidad de datos de entrenamiento

  Es la cantidad de ejemplos reales de algún evento o fenómeno que se utilizará para el entrenamiento de un modelo de *Machine Learning*. Se calcula con un conteo simple.

  Indicador: Vector de Características

  En *Machine Learning*, una característica es una propiedad individual medible de un evento o un fenómeno. Un conjunto de características forman un patrón que se utilizará en el entrenamiento de un modelo de *Machine Learning*. Este patrón toma un valor numérico y se representa mediante un vector, también llamado "Vector de características".

- **Indicadores de la variable dependiente:**

  Indicador: Número de verdaderos positivos

  Corresponde al número de predicciones, tal que, la predicción es positiva y el evento real también. En nuestro caso sería el número de veces que el sistema detecta correctamente una avalancha. Se calcula con un conteo simple.

Indicador: Número de verdaderos negativos

Corresponde al número de predicciones, tal que, la predicción es negativa y el evento real también. En nuestro caso sería el número de veces que el sistema clasifica correctamente un evento que no es una avalancha. Se calcula con un conteo simple.

Indicador: Número de falsos positivos

Corresponde al número de predicciones, tal que, la predicción es positiva pero el evento real es negativo. En nuestro caso sería el número de veces que el sistema detecta una avalancha cuando en la realidad no ha sucedido ninguna. Se calcula con un conteo simple.

Indicador: Número de falsos negativos

Corresponde al número de predicciones, tal que, la predicción es negativa pero el evento real es positivo. En nuestro caso sería el número de veces que el sistema clasifica un evento como una no-avalancha cuando en la realidad si ha sucedido una. Se calcula con un conteo simple.

1.7. Alcances

Con el desarrollo de la presente tesis se logrará:

- Recolectar data sobre avalanchas.
- Detectar avalanchas en un rango que cubre hasta aproximadamente 3Km a la redonda desde donde esta ubicado el sensor [9].
- Diseñar un sistema de monitoreo y detección de avalanchas en la laguna Palcacocha al que se pueda acceder de forma remota desde una computadora en la sede del INAIGEM en la ciudad de Huaraz, la implementación de este sistema cuenta como parte de la propuesta.

1.8. Limitaciones

Las limitaciones con respecto al contexto del trabajo de tesis son los siguientes:

- El lugar donde se llevará a cabo la investigación, la laguna Palcacocha, es una zona de complicada accesibilidad.

- Todos los estudios respecto a la detección de avalanchas se llevarán a cabo solo en la laguna de Palcacocha.

Las limitaciones con respecto a la disponibilidad de recursos:

- Debido a limitaciones económicas se trabajará con un solo sensor.
- El hecho de trabajar con un solo sensor resulta en la limitación de que no será posible detectar la ubicación exacta de la avalancha ni tampoco nos permitirá clasificar la intensidad de las avalanchas detectadas.

Las limitaciones con respecto a los antecedentes son los siguientes:

- Aunque existen trabajos en la literatura con respecto a la detección de avalanchas utilizando un único sensor, estos son muy pocos.
- No existen antecedentes en la zona de estudio respecto al tema de detección de avalanchas con sensores infrasónicos.

II. Marco teórico

2.1. Conceptos Previos

2.1.1. Infrasonido

El sonido es una onda mecánica generada cuando un objeto produce variaciones de presión acústica, por ejemplo esto sucede cuando un objeto vibra, estas vibraciones hacen que las partículas alrededor del objeto (normalmente aire) se muevan también en un movimiento vibratorio, de esta forma transportando energía a través del medio. La frecuencia del sonido es el número de veces por segundo que las partículas se mueven un ciclo alrededor de un punto fijo (medido en Hz). Los humanos pueden escuchar los sonidos en el rango desde los 20 Hz hasta los 20000 Hz, cualquier frecuencia fuera de este rango es inaudible para los humanos, tomando como referencia este rango es que se clasifican los sonidos como Ultrasonidos cuando son de frecuencia mayores a 20000 Hz e Infrasonidos cuando están por debajo de los 20 Hz.

La figura siguiente representa el coeficiente de absorción del sonido en distintos materiales, se observa que a más bajas frecuencias más bajo es el coeficiente de absorción para los materiales examinados.

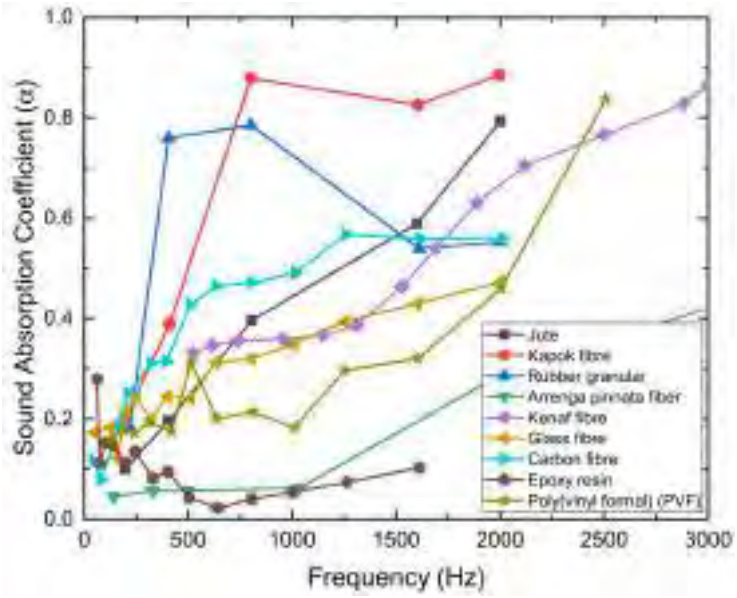

Figura 6: Coeficiente de absorción del sonido en distintos materiales

Fuente: Kumar, Sanjay, y Heow Pueh Lee, "The present and future role of acoustic metamaterials for architectural and urban noise mitigations"

Esta demostrado que el infrasonido tiene un coeficiente de atenuación más bajo que los sonidos de alta frecuencia [20]. Por tanto, los infrasonidos pueden viajar distancias más grandes desde la fuente del sonido, específicamente se sabe que los infrasonidos producidos por una avalancha pueden viajar hasta 14 Km [7] (aunque para su óptima detección, se recomienda que los sensores de infrasonido se encuentren a una distancia menor a 3Km de la fuente donde se produce las avalanchas [9]).

2.1.2. Avalanchas

Las avalanchas son movimientos de masa que descienden de forma rápida o súbita a través de una pendiente, montaña o glaciar. Las avalanchas están compuestas de nieve pero también pueden contener rocas, tierra o hielo y pueden alcanzar una velocidad de hasta 60 m/s [21]. Las avalanchas se generan debido a muchos factores, una de ellas es la pendiente de la montaña donde se encuentra la nieve, las avalanchas son originadas desde montañas con pendientes entre 25° y 55° [22], las más comunes siendo entre 30° y 45° [23], otros factores son la acumulación de la nieve y las condiciones meteorológicas de la zona, la superficie de la nieve esta siendo constantemente afectada por procesos térmicos y mecánicos, en particular

por el viento, el cual puede generar capas de placa inestables que son propensas a ocasionar su desprendimiento [23]. Todos estos factores condicionan la cantidad de nieve que puede ser desplazada y este, a su vez, condiciona el comportamiento y evolución de la avalancha a lo largo de su trayectoria.

Las avalanchas pueden clasificarse, por la forma en que se originan, como avalanchas de nieve suelta o como avalanchas de placa [23]. Las avalanchas de nieve suelta se originan desde un cierto punto en la ladera y van incrementando su tamaño con la nieve que se va adhiriendo en su trayectoria, las avalanchas de placa son originadas debido a la existencia de capas de nieve que no se encuentran bien cohesionadas a la cubierta glaciar, de esta forma movilizando todo un bloque de nieve (con forma de placa) al mismo tiempo cuando se produce la ruptura y generando una avalancha. Las avalanchas de placa pueden ser clasificadas a su vez como húmedas o secas, cabe indicar, que este tipo de avalanchas usualmente son mucho más peligrosas que las avalanchas de nieve suelta debido a la gran magnitud que pueden alcanzar.

Las avalanchas pueden clasificarse también por su magnitud, aunque distintas instituciones usan distintas clasificaciones. La información de la tabla que se mostrará a continuación fue definida por la institución "*The European Avalanche Warning Services*" (EAWS) y basada en la clasificación canadiense:

2.1.3. Avalanchas como fuentes de ondas infrasónicas

Las primeras investigaciones sobre señales acústicas relacionadas con avalanchas van hasta los años 1970 [24] [25], cuando se observaron emisiones de señales acústicas de baja frecuencia en la cubierta de nieve justo antes de que sucediera una avalancha natural. Posteriores estudios demostraron también que cuando sucedía un evento de avalancha se producían fuertes vibraciones en el espectro del infrasonido [15], estas vibraciones infrasónicas se propagan grandes distancias y pueden ser utilizadas en la detección remota de estos fenómenos.

Tamaño	Daño Potencial	Distancia máxima recorrida	Dimensiones	
			Longitud trayectoria	Volumen
Pequeña	Improbable de enterrar a una persona. En terrenos extremos, existe el riesgo de caídas.	Se detiene en el área donde se origina (en la pendiente).	< 50m	100 m3
Media	Puede enterrar, herir y matar personas.	Se detiene al final de la pendiente.	50 - 200m	1000 m3
Grande	Puede enterrar y destruir autos, pequeñas edificaciones y árboles.	Llega hasta superficies planas (menores a 30°) y pueden continuar adelante por una distancia menor a 50 m.	< 1000 m	10 000 m3
Muy Grande	Puede destruir edificaciones y pequeños bosques.	Llega hasta superficies planas (menores a 30°) y pueden continuar adelante por una distancia mayor a 50 m.	< 2000 m	100 000 m3
Extremadamente Grande	Potencial destructivo catastrófico.	Alcanza el piso del valle.	> 2000 m	>100 000 m3

Tabla 1: Clasificación de avalanchas por su magnitud

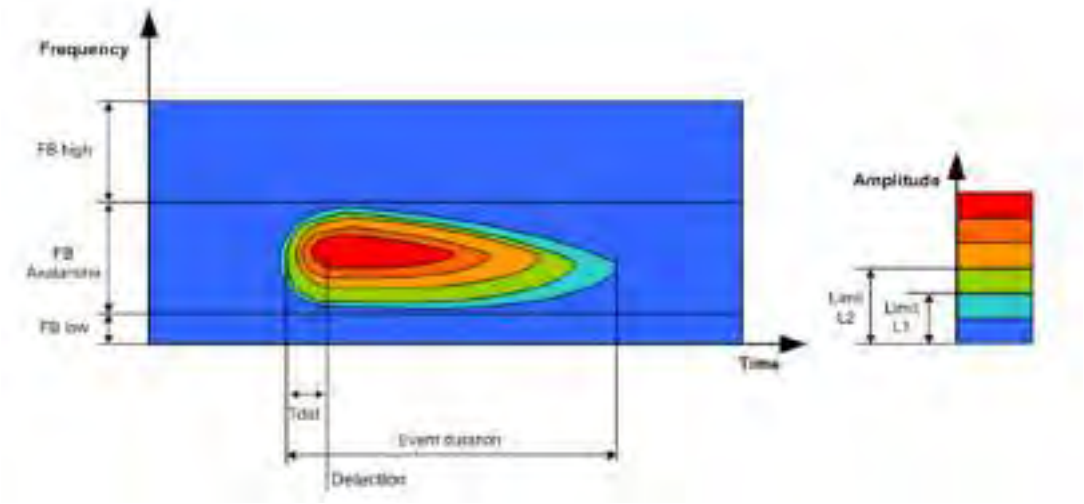

Figura 7: Ilustración de un evento de avalancha representado en el espectrograma de este evento en el infrasonido

Fuente: [12]

Uno de los primeros estudios relacionados al tema fue realizado por el Laboratorio de Electromagnetismo y Acústica de la Escuela Politécnica Federal de Zúrich en 1998, en este estudio se demuestra empíricamente la relación de las avalanchas y el infrasonido. Para

su estudio diseñaron y desarrollaron un goniómetro compuesto de un arreglo de sensores de infrasonido, para el experimento generaron avalanchas artificiales mediante explosiones inducidas [15], en la figura siguiente se puede observar el espectrograma de las señales acústicas correspondientes a una serie de avalanchas, las cuales demuestran la presencia de señales de frecuencias inferiores a los 10 Hz.

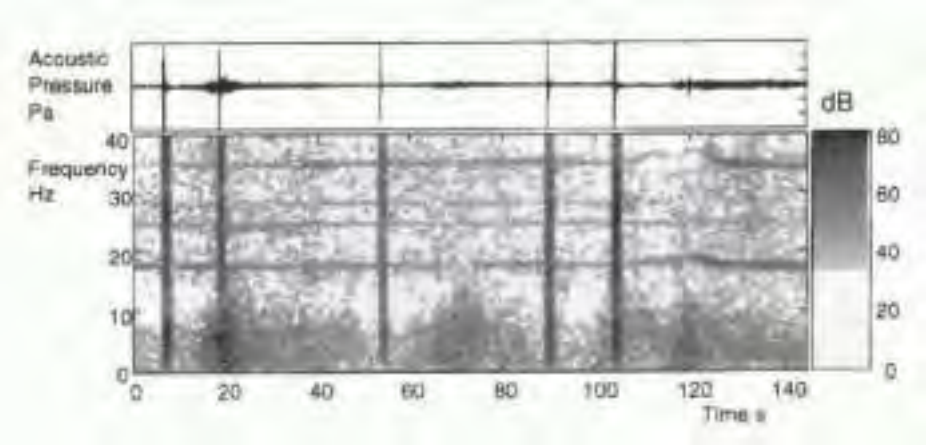

Figura 8: Espectrograma correspondiente a una serie de explosiones (líneas verticales) y avalanchas (áreas grises por debajo de los 10 Hz). Las líneas horizontales (de 17 Hz y sus armónicos) corresponden a la presencia de un helicóptero.

Fuente: [15]

Diferentes tipos de avalanchas emiten sonidos con picos de frecuencia en distintos rangos, aunque se había estudiado que típicamente las avalanchas de nieve se encuentran dentro de la banda de 0.8 a 12Hz [26], un estudio encontró que algunas avalanchas pequeñas de nieve suelta producen picos de sonidos en el límite superior del infrasonido, mientras que las avalanchas de placa producen picos de infrasonido en el rango de 1 a 3 Hz [27], otros estudios mostraron que muchas avalanchas emiten sonidos entre 2 y 8 Hz en promedio en toda la duración de la avalancha, con la mayor concentración de energía observada al inicio de las avalanchas con picos de frecuencia entre 1 y 5 Hz, cuya etapa esta relacionada a la nube de polvo generada en la etapa inicial de la avalancha; mientras que en etapas posteriores se observó menor energía diseminada en toda la banda de 1 a 40 Hz, cuya etapa esta relacionada al comportamiento posterior de la avalancha donde se convierte en un flujo de alta densidad mientras va acumulando nieve húmeda a mas bajas altitudes [28]; también cabria añadir que las avalanchas en su etapa posterior podrían no solo

arrastrar nieve, sino también rocas, hielo y tierra, cambiando su comportamiento y añadiendo sonidos propios de estos materiales a los que producen las avalanchas de nieve. Por esta razón es que cuando se trata de detección de avalanchas, muchos estudios se enfocan en la detección de la etapa inicial de la avalancha cuyo comportamiento es más general, de una duración mas corta y además de que es más útil en la implementación de un sistema de detección temprana.

2.2. Sensores de Sonido e Infrasonido

Un sensor es un dispositivo electrónico que puede convertir variaciones del entrono físico del mundo real en impulsos eléctricos. Existen distintos tipos de sensores para detectar distintos tipos de eventos y señales del mundo real. Los sensores de sonido pueden detectar variaciones en la vibración del presión acústica en el aire, estos son más conocidos por el nombre de "micrófonos".

Existen distintos tipo de micrófonos, que por su construcción pueden presentar mejores capacidades unos de otros, la gran mayoría de ellos están compuestos de un diafragma que vibra al ritmo de los sonidos y de esta forma convertirlos en corrientes eléctricas. En la actualidad, gran parte de los micrófonos comerciales utilizan cambio de capacitancia (micrófonos de condensador), piezoelectricidad (micrófonos piezoeléctricos), o inducción electromagnética (micrófonos dinámicos), los micrófonos comerciales están diseñados para trabajar con los sonidos audibles a los humanos, es decir trabajan en un rango aproximad desde los 80 Hz hasta los 20 KHz. Existen otro tipo de micrófonos que pueden trabajar en bandas del sonido no comunes como la banda del infrasonido, estos son llamados microbarógrafos.

2.2.1. Microbarógrafos

Los microbarógrafos o microbarómetros son sensores que pueden medir la presión del aire, se diferencian de un barómetro común en que estos son mucho más sensibles a las variaciones de presión. Estos sensores están construidos con componentes piezoresistivos micromecanizados, los cuales permiten que se pueda medir las fluctuaciones de presión de un orden muy pequeño, con una resolución de pascales (Pa) o microbares (μbar). Los transductores de presión piezorresistivos (*Piezoresistivity pressure transducers* o PPTs),

fabricados con tecnología de sistemas microelectromecánicos (*microelectromechanical systems* o MEMS), se utilizan ampliamente en aplicaciones industriales y médicas para mediciones de baja presión. En los TPPs MEMS, los materiales que cambian de resistencia bajo tensión aplicada se conectan a diafragmas micromecanizados a base de silicio. La deflexión del diafragma y el correspondiente cambio de resistencia se miden mediante un puente de *Wheatstone*. Las PPT MEMS pueden detectar presiones estáticas y dinámicas con gran precisión. Es por ello que los TPP MEMS a frecuencias muy bajas pueden ser utilizados como elemento sensor diferencial para detectar señales de infrasonidos siempre y cuando dicho sensor tenga la capacidad de rechazar la presión estática que puede saturar el diafragma y hacerlos insensibles a las presiones en la banda de frecuencias objetivo, para lo cuál este tipo de sensores suelen estar equipados con filtros neumáticos.[29].

Una desventaja de su gran sensibilidad es que son muy propensos a algunos movimientos de partículas de aire como el viento. Este y otros tipos de ruidos se describen a continuación.

2.2.1.1. Ruidos que afectan la medición del infrasonido

El ruido es un tipo de señal que estorba o entra en conflicto con la medición de la señal que nos interesa, algunos tipos de ruido que afectan la medición del infrasonido en los microbarógrafos son:

- Tipo electrónico, como el ruido de disparo o el ruido térmico, este usualmente se da debido al movimiento de electrones y es inherente a los componentes electrónicos.
- Tipo electromagnético, como por ejemplo las ondas de wifi o la interferencia electromagnética que emite alrededor de sí cualquier cable que este transmitiendo energía de alta frecuencia.
- Tipo acústico, que son vibraciones acústicas originados por objetos que se mueven o vibran a una frecuencia cercana de la señal que se quiere medir. Fuentes de ruido que afectan la banda de infrasonido pueden ser: motores, vehículos moviéndose, aviones o helicópteros. Otro tipo de ruido acústico es el viento, el cuál es de naturaleza muy distinta a los otros ruidos acústicos mencionados anteriormente. La naturaleza del viento es aleatoria y no coherente, es decir no proviene de un lugar fijo ni de una fuente definida. El ruido producido por el viento afecta más a las señales de las más bajas frecuencias de la banda del infrasonido, aunque también puede afectar todo el

rango de esta banda si la velocidad del viento es lo suficientemente alta.

2.3. Procesamiento Digital de Señales

El procesamiento de señales se refiere al procedimiento mediante el cuál las señales pueden ser medidas, analizadas y manipuladas. Las señales usualmente se encuentran de forma analógica en la naturaleza, este es el caso del sonido que como se explicó anteriormente es una onda mecánica, las señales analógicas pueden ser procesadas en su forma analógica por medio de circuitos integrados, transistores, filtros analógicos, entre otros sistemas analógicos, este tipo de procesamiento tiene como desventaja de que el procedimiento requiere de circuitos electrónicos físicos y por lo tanto mientras más complejo es el procesamiento más grandes y complejos se hacen estos circuitos. Otra forma de procesamiento es el procesamiento digital de señales, este consiste en convertir una señal analógica en digital, para que de esta forma, la señal puede ser analizada en forma de bits por medio de un computadora o un microprocesador, este tipo de procesamiento tiene la ventaja de que es fácil modificar las operaciones mediante cambios en el software, a diferencia de un procesamiento analógico donde las operaciones se encuentran en circuitos electrónicos y la reconfiguración es mucho más complicada.

2.3.1. Conversión Analógica-Digital

Este es el procedimiento mediante el cual se transforma una señal analógica en digital, este se realiza mediante circuitos que se llaman "convertidores A/D" (ó ADC). La conversión A/D es un proceso de tres pasos, los cuales son:

- Muestreo: El muestreo consiste en la toma de una secuencia de muestras discontinuas de una señal analógica (señal continua). El número de muestras que se toman de una señal analógica (también llamada señal muestreada) por segundo es lo que se conoce como tasa de muestreo. La tasa de muestreo puede ser medida en "muestras/segundo" pero también se suele utilizar los "Hz". La resolución de una señal digital en el tiempo esta definida por su tasa de muestreo, es decir, si la tasa es mayor, la resolución también es mejor. El número de muestras que resultan del muestreo de una señal analógica esta determinado por su tasa de muestreo, por ejemplo, si se tiene una señal de longitud

t segundos y se muestrea a una tasa de muestreo es de f Hz, el número de puntos o muestras resultantes de esta señal serían $t * f$.

- Cuantificación: Después de haber hecho el muestreo a la señal analógica lo que se obtiene es una señal discreta, una señal discreta es una señal representada en el tiempo por puntos o muestras discontinuas pero cuya amplitud aún es continua, el procedimiento mediante el cuál se convierte una señal discreta en una señal digital se llama cuantificación, este consiste en expresar cada valor de muestra de la amplitud como un número finito de dígitos (en lugar de infinito).

- Codificación: El proceso de codificación consiste representar cada valor discreto de la señal digital como una secuencia binaria de bits.

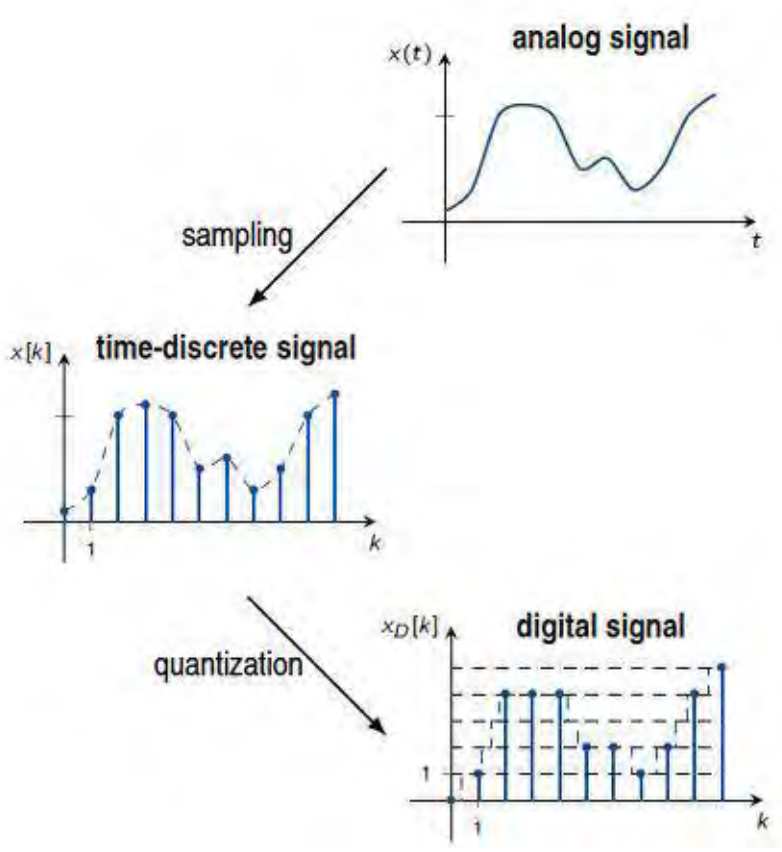

Figura 9: Proceso de digitalización (muestreo y cuantificación) de una señal analógica.

Fuente:

https://static.javatpoint.com/tutorial/dip/images/analog-signals-to-digital-signals2.png

2.3.2. Teorema de Muestreo de Nyquist-Shannon

Para saber cual es el valor ideal de tasa de muestreo para una señal, es importante conocer el Teorema de muestreo de Nyquist-Shannon. Este es un teorema fundamental que expresa la relación entre señales analógicas y las señales digitales, el cuál indica que, si la frecuencia mas alta de una señal es f_{max} entonces su tasa de muestreo debe ser mayor o igual a $2f_{max}$, en caso no se cumpla dicha condición, los elementos de las frecuencias más altas no serán correctamente representados, lo que produce el fenómeno conocido como solapamiento o *aliasing*.

2.3.3. Filtros Electrónicos

Un filtro electrónico es un elemento que altera las características de amplitud y/o de fase de una señal con respecto a la frecuencia. Idealmente, un filtro no añade nuevas frecuencias a la señal de entrada, pero su uso puede cambiar las amplitudes relativas de los distintos componentes de frecuencia y/o sus relaciones de fase. Los filtros son usualmente usados en electrónica para enfatizar señales en ciertos rangos de frecuencia y rechazar señales en otros rangos. De esta forma un filtro tiene una ganancia, la cual es dependiente de la frecuencia de una señal. Existen distintos tipos en los que se pueden clasificar los filtros, en la sección 2.3.3.1 describiremos los tipos de filtros según su respuesta en frecuencia.

2.3.3.1. Tipos de filtro por su Respuesta en Frecuencia

Según su respuesta en frecuencia los filtros se pueden clasificar en cuatro tipos:

- Filtro Pasabajos: Este tipo de filtros tiene la función de dejar pasar todas las frecuencias de una señal que se encuentren por debajo o sean inferiores a una cierta frecuencia establecida, esta frecuencia se denomina "frecuencia de cortez su valor se elige en la etapa de diseño del filtro.

- Filtro Pasaaltos: Este tipo de filtros tiene la función de dejar pasar todas las frecuencias de una señal que se encuentren por encima o sean superiores a una cierta frecuencia establecida, esta frecuencia se denomina "frecuencia de cortez su valor se elige en la etapa de diseño del filtro.

- Fitro Pasabanda: Este tipo de filtros tiene la función de dejar pasar una banda específica

de frecuencia de una señal mientras atenúa las demás bandas, en este sentido para el diseño de un filtro pasabanda se requiere de elegir dos frecuencias de corte, una inferior y una superior.

- Fitro Notch o Rechaza Banda: Este tipo de filtros tiene la función de rechazar una banda específica de frecuencia de una señal mientras deja pasar las demás bandas, en este sentido para el diseño de un filtro rechaza banda se requiere de elegir dos frecuencias de corte, una inferior y una superior.

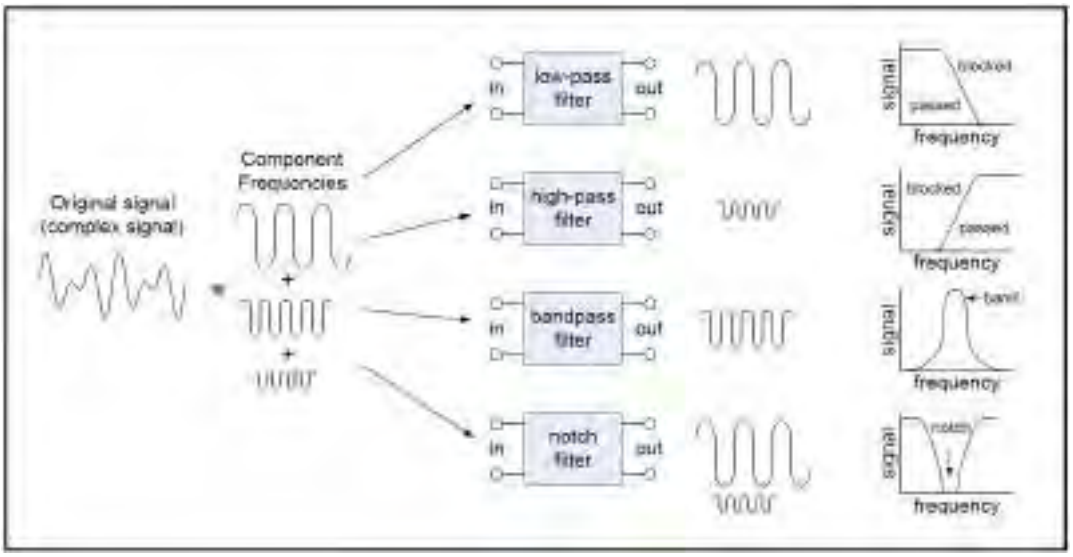

Figura 10: Los 4 tipos de filtros más importantes según su respuesta en frecuencia.
Fuente: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-filters/

2.3.3.2. Función de transferencia de un filtro

El diseño de filtros electrónicos involucra el diseño de la relación entre su salida y su entrada, siendo la entrada la señal que se quiere filtrar y la salida la señal modificada por el filtro, esta relación puede ser representada matemáticamente por una función que se denomina "Función de transferencia". La función de transferencia usualmente se representa como la transformada de Laplace de la señal de salida entre la transformada de la señal de entrada. A continuación se representa función de transferencia H(s) como la relación entre la señal de salida Y(s) y la señal de entrada X(s), todos en función de la variable compleja s:

$$H(s) = \frac{Y(s)}{X(s)} \tag{1}$$

Donde $s = \sigma + j\omega$.

Las raíces del polinomio en el denominador X(s) se conocen como “polos” y las raíces en el numerador Y(s) se conocen como “ceros”, el grado de los polinomios en el denominador determina la “orden” para los filtros de uso más común. En la ecuación 2 se muestran las funciones de transferencia de distintos tipos de filtros de "segundo orden", se denominan así porque no existen elementos en dichas funciones de transferencia con un grado mayor a 2.

$$\begin{cases} Pasabajo : H(s) = \frac{K}{s^2+bs+a} \\ Pasabanda : H(s) = \frac{Ks}{s^2+bs+a} \\ Pasaalto : H(s) = \frac{Ks^2}{s^2+bs+a} \end{cases} \tag{2}$$

El orden de los filtros determina cuánto se puede acercar un filtro a su respuesta ideal, por ejemplo, en la figura 11 se observa un filtro Butterworth Pasabajos cuya frecuencia de corte es 1 Hz, en este filtro, la respuesta ideal sería la de eliminar todos los componentes superiores a 1 Hz de modo que la función de transferencia resulte en la forma de una escalera, sin embargo, en la práctica se da que el filtro no es ideal y va atenuando progresivamente las frecuencias por encima de la frecuencia de corte, mientras más alto sea el orden del filtro más componentes no deseadas serán filtradas y más cerca estará de su comportamiento ideal. Una desventaja de diseñar filtros de mayor orden es que son más complejos y su implementación tienen más costo computacional.

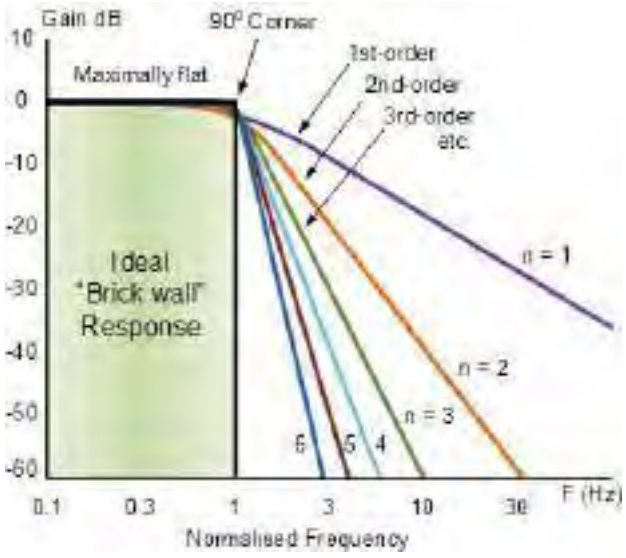

Figura 11: Respuesta en frecuencia de un filtro Butterworth Pasabajos para distintas ordenes del mismo, la frecuencia de corte del filtro es 1 Hz.

Fuente: https://www.electronics-tutorials.ws/filter/filter_8.html

2.3.3.3. Filtro de Puerta de Ruido Espectral

Un filtro de Puerta de Ruido o "*Noise Gate*" es un tipo de filtro que se utiliza en la limpieza de ruido de señales de audio, como su nombre indica, consiste en una compuerta o puerta que solo permite pasar las señales que superen cierto umbral prefijado, es bastante utilizado para eliminar el ruido durante las pausas. Además del umbral (*Threshold*), en las puertas de ruido se pueden configurar otros parámetros como son el tiempo de ataque (*Attack Time*), que es el tiempo que se demora la puerta de ruido para abrirse luego que la señal sobrepase el umbral, y el tiempo de relajación (*Release Time*), que es el tiempo que se demora la puerta de ruido para cerrarse luego que la señal caiga por debajo del umbral.

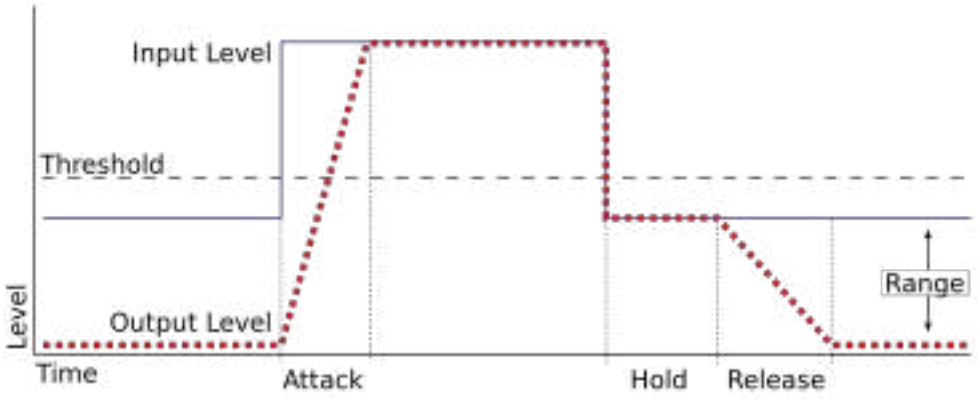

Figura 12: Ejemplo de una Puerta de Ruido con el umbral, tiempo de ataque y relajación definidos.

Fuente: https://en.wikipedia.org/wiki/Noise_gate

Un tipo especial de filtro de Puerta de Ruido es el llamado Filtro de Puerta de Ruido Espectral o "*Spectral Gating Noise Filter*", el cuál es un tipo de filtro muy novedoso que se ha estado estudiando en los últimos años y que ha sido adoptado por algunos *softwares* de edición de sonido como *Audacity*, a diferencia de las puertas de ruido comunes, los filtros de Puerta de Ruido Espectral trabajan a nivel espectral o en el dominio de la frecuencia. Aunque existen varios tipos de algoritmos en la literatura con los se puede implementar un Filtro de Puerta de Ruido Espectral, uno de los más novedosos es el que desarrollo Tim Sainburg y su equipo de investigación para la Universidad de California [30][31], el cuál esta basado en el algoritmo del *software Audacity*. Dicho algoritmo se describe a continuación.

Dada una forma de onda de audio que contiene una señal de sonido con ruido de fondo incorporado (S_n) y una pequeña muestra de audio de la misma o similar forma de onda que contenga solo el ruido de fondo (N), se describen a continuación los pasos que sigue el algoritmo:

- Calcular la Transformada de Fourier de Tiempo Corto de N ($spec_n$).
- Calcular la media y la desviación estándar de $spec_n$ para cada componente de frecuencia sobre el tiempo.
- Calcular la Transformada de Fourier de Tiempo Corto de S_n ($spec_s$).

- Por cada componente de frecuencia, calcular un umbral de nivel de ruido basado sobre la media y la desviación estándar de $spec_n$.
- Generar una máscara a partir de $spec_s$, basada en la potencia de $spec_s$ y los umbrales hallados de $spec_n$.
- Suavizar la máscara en frecuencia y tiempo.
- Aplicar la máscara a $spec_s$ para remover el ruido.
- Calcular la Transformada Inversa de Fourier de Tiempo Corto de $spec_s$ para generar la señal sin ruido en el dominio del tiempo.

2.3.4. Segmentación y Enventanado

En el procesamiento digital de señales, la segmentación y el enventanado son dos técnicas que se suelen utilizar para el análisis de señales de audio o señales ECG. Se detallará a continuación el funcionamiento de ambas técnicas.

2.3.4.1. Segmentación o Framing

La segmentación es la separación de una señal en pequeños pedazos de igual tamaño llamados segmentos o *frames*. La segmentación se hace para poder revelar características no aparentes de la señal fuente y para poder realizar un análisis de frecuencia en función del tiempo. Por ejemplo, se sabe que el análisis de frecuencia para señales discretas y digitales se hace a través de la transformada rápida de Fourier (*Fast Fourier Tranform* o FFT).

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \quad 0 \leq k \leq N-1 \tag{3}$$

Donde k es la posición de los puntos discretos de la transformada, k puede tomar hasta N valores, es decir, la cantidad de puntos que tiene la FFT de una señal es igual a la cantidad de puntos (muestras) de esa señal discreta.

Por tanto, si deseamos aplicar la FFT a una señal digital para poder obtener sus características en frecuencia solo podríamos obtener este análisis para TODA la señal y perderíamos las características temporales de la señal en el dominio de la frecuencia, es aquí donde nos es útil la segmentación de *frames* de la señal, de esta forma si separamos

nuestra señal de análisis en segmentos más pequeños y le aplicamos la FFT a cada uno de estos segmentos, podemos obtener un análisis en frecuencia y al mismo tiempo mantener las características temporales de dicha señal. Cuando se quiere elegir el tamaño del *frame*, se debe tener en cuenta estos aspectos: Mientras más pequeña la longitud del *frame*, mejor es la resolución temporal de la señal segmentada pero se pierde fidelidad en el análisis de frecuencia de cada segmento, y de forma inversa cuanto más grande sea la longitud del *frame*. Para procesamiento de señales es común utilizar un tamaño de *frame* que sea una potencia de 2, ya que esto disminuye la carga computacional del algoritmo FFT. Para procesamiento de audio en general se usa valores que rondan entre las 256 y 8192 muestras, esta cantidad esta influenciada también por la tasa de muestreo. Por ejemplo si se utiliza un tamaño de *frame* de 512 muestras para una señal de audio común que normalmente se encuentra a 44.1 KHz (44100 muestras/segundo), haciendo una división simple, cada *frame* correspondería a 11.6 ms en duración de tiempo, esta longitud de *frame* tan pequeño es útil para clasificación de vocablos en señales de audio, ya que ese es el tiempo promedio de duración en que un ser humano vocaliza una letra, la longitud de los *frames* deberá diseñarse según las características de cada aplicación.

La segmentación de la señal es necesaria para distintos procedimientos donde se quiera analizar la frecuencia en función del tiempo, por ejemplo, en los espectrogramas se grafican las propiedades de frecuencia de las señales en función del tiempo con una estructura de mapas de calor. Aún con las ventajas que se obtienen, segmentar la señal también ocasiona un problema que explicaremos a continuación.

Cuando aplicamos la Transformada Discreta de Fourier a una señal finita, se asume que este es un periodo o un número entero de periodos de una señal más grande que se extiende de forma infinita. Al separar nuestra señal en pequeños segmentos temporales, cada segmento individual se convierte en una nueva señal, si queremos aplicar la FFT a este segmento, entonces para una correcta aplicación de la FFT, este segmento deberá contener un periodo o un número entero de periodos de una señal más grande infinita. Las señales obtenidas por el sensor en la vida real son no-estacionarias y cambian constantemente de propiedades y sus segmentos se comportan de igual forma, por tanto, aunque cada segmento no forma realmente parte de una señal periódica infinita, debemos asumir que si para poder aplicar la FFT. Esto ocasiona que los bordes de cada segmento sean vistos como

discontinuidades (ver figura 13). Si aplicamos la FFT a segmentos con discontinuidades, entonces se generan componentes de alta frecuencia no deseados en la transformada de Fourier resultante; esto es lo que se conoce como Manchado Espectral o *Spectral Leakage*.

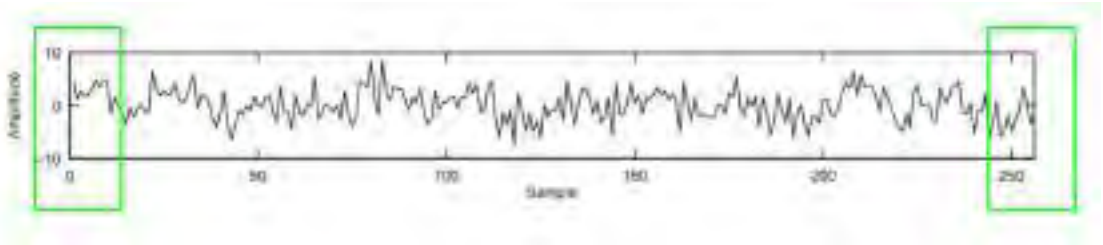

Figura 13: Segmento de una señal, los bordes son vistos como discontinuidades.
Fuente: https://github.com/musikalkemist/AudioSignalProcessingForML/

Para poder superar los problemas derivados de las discontinuidades se aplican "ventanas" o "*windows*". Esta es la técnica conocida, en procesamiento de señales, como "Enventanado" o "*windowing*".

2.3.4.2. Enventanado o Windowing

La aplicación de ventanas en cada segmento reduce el impacto de la segmentación en las propiedades estadísticas de la señal y soluciona los problemas derivados de las discontinuidades que se generan. "Enventanar" una señal implica suavizarla en los bordes, de modo que esta tienda a valores de amplitud cero en los bordes, de esta forma, las discontinuidades en los bordes se vuelven invisibles.

La aplicación de una ventana en una señal es una multiplicación simple como se expresa en la ecuación 4, donde, dada una señal $X[k]$ definida para todos los k, y una función de ventana $W[k]$ definida en un rango limitado $k \in [0, L)$, donde L es la longitud del segmento, podemos extraer la ventana de una señal como:

$$X_W[k] = X[k]W[k] \tag{4}$$

Una clásica ventana que se suele usar es la ventana de tipo Hann cuya función es:

$$W[k] = (sin(\pi k/L))^2 \quad k \in [0, L) \tag{5}$$

En la figura 14 se ilustra el procedimiento de "Enventanado" en un segmento de L=250,

con una ventana de tipo Hann.

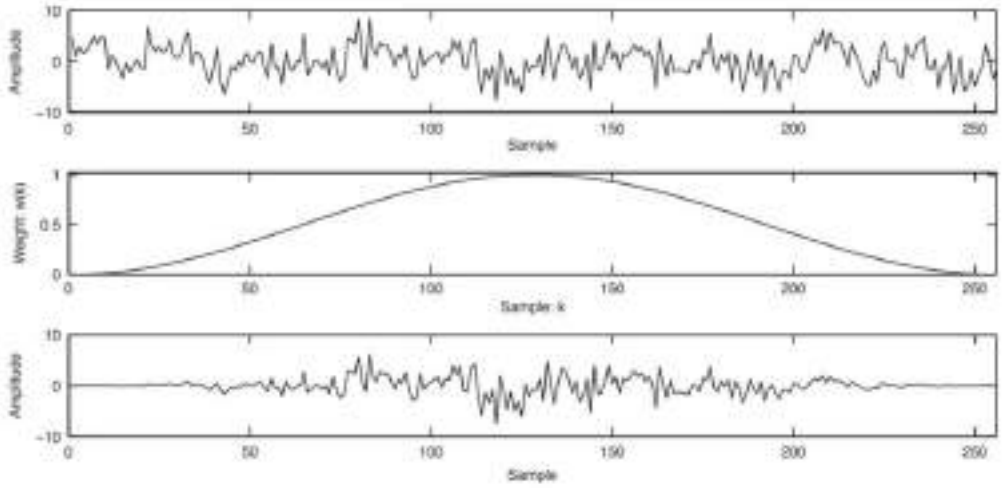

Figura 14: La primera imagen es el segmento de una señal, la segunda una ventana Hann que será aplicada al segmento mediante una multiplicación simple, y la última imagen corresponde a la señal “enventanada”.

Fuente: https://github.com/musikalkemist/AudioSignalProcessingForML/

Notar que si aplicamos las ventanas a cada segmento, la ventana suavizara los bordes de cada segmento, llevándolos a cero y provocando que perdamos información en los bordes de cada segmento. Para solucionar esto, los segmentos y su posterior *windowing* deben tener lo que se conoce como “solapamiento”.

El “Solapamiento” simplemente implica que la segmentación de la señal que se diseñó originalmente ahora se realizará de forma que cada segmento puedan solaparse entre sí, el procedimiento de Enventanado se dará de igual forma en cada segmento. Realizando este procedimiento, lograremos que, aunque se pierda información en los bordes de un segmento, esa información pueda ser rescatada en el siguiente segmento, una elección usual de solapamiento se encuentra entre el 25 % y el 50 %. Todo el proceso de Segmentación (*Framing*) y de Enventanado (*Windowing*) con solapamiento se ilustra en la siguiente figura:

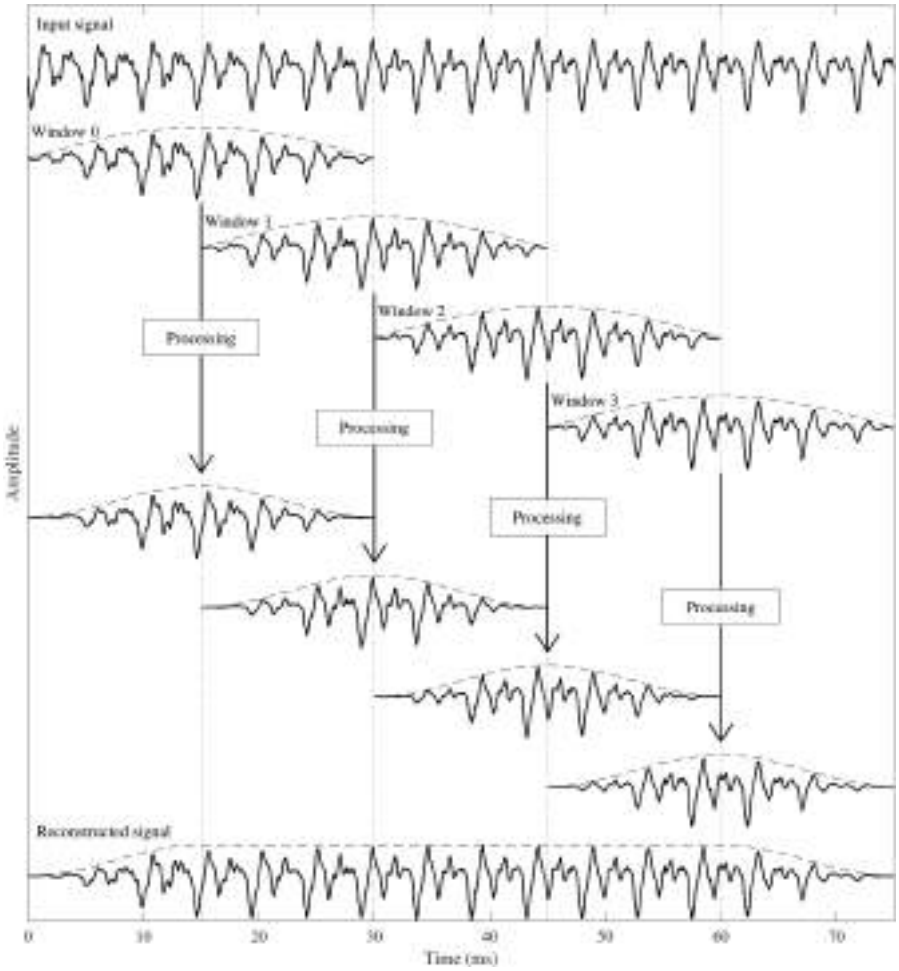

Figura 15: Proceso completo de segmentación y enventanado con solapamiento del 50 %.

Fuente: [32]

2.4. Machine Learning y Aprendizaje Supervisado

El *Machine Learning* es una rama de la inteligencia artificial, es un área de estudio que ha estado con nosotros ya por varias décadas pero cuyas aplicaciones solo se han vuelto mas accesibles al público común en años recientes.

En 1959, Arthur Samuel, un pionero en *Machine Learning* (Aprendizaje de Maquina o Aprendizaje Automático en español) lo definió de la siguiente forma: "[El *Machine Learning*] es el campo de estudio que les otorga a las computadoras la habilidad de aprender sin haber sido explícitamente programadas". Se podría decir entonces que el *Machine Learning* es un tipo de IA que ya no depende de unas reglas pre-definidas y un programador, sino que la computadora puede establecer sus propias reglas y aprender por si

misma [33]. Este proceso de aprendizaje se da entregándole a la computadora una gran cantidad de datos de la cual pueda "aprender" y "entrenarse, el resultado de este entrenamiento es la obtención de un modelo con el cual podremos realizar cierta tarea determinada o hacer predicciones de determinados eventos. También existen varios algoritmos, que dependiendo de la tarea que se desee realizar, será mas adecuado trabajar con uno u otro.

Uno de los tipos de *Machine Learning* es el Aprendizaje Supervisado, que como su nombre indica, consiste en realizar el entrenamiento con supervisión humana, en general este algoritmo se puede dividir en dos fases: La fase de entrenamiento y la fase de evaluación.

La **fase de entrenamiento** consiste en alimentar el algoritmo con data que incluye los datos de entrada y sus etiquetas con la salida o solución esperada ("*labeled data*"), estos se organizan en un set de datos (o dataset), de esta forma se le "enseña" al algoritmo a reconocer los patrones de dicha data y cual etiqueta le correspondería a dichos patrones. Este dataset es también llamado Set de Entrenamiento o *Training Set*, el resultado que se obtiene de la fase de entrenamiento es un modelo de *Machine Learning*.

Figura 16: Un Dataset de entrenamiento etiquetado ("*Labeled Dataset*") para entrenamiento supervisado.

Fuente: [33]

La **fase de evaluación** se efectúa una vez se ha obtenido el modelo en la anterior fase, esta fase consiste en proporcionar al algoritmo un set de datos no etiquetados distintos a los que se usaron para entrenar el modelo, de esta forma, el modelo se encargará de etiquetar

este nuevo set de datos con lo que ha "aprendido" de la fase de entrenamiento. Conociendo las verdaderas etiquetas, es posible obtener un análisis del rendimiento o efectividad de este modelo. Tomemos como ejemplo un modelo que se ha entrenado bajo eventos que tienen dos tipos de etiquetas: positivos (por ejemplo, avalanchas) y negativos (por ejemplo, no avalanchas/ruidos), para analizar el rendimiento podemos utilizar los cuatro parámetros que se describirán a continuación:

- Verdaderos Positivos (VP): Es un tipo de acierto en el cual el modelo ha etiquetado un evento como "positivo" y dicho evento si es positivo.
- Verdaderos Negativos (VN): Es un tipo de acierto en el cual el modelo ha etiquetado un evento como "negativo" y dicho evento si es negativo.
- Falso Negativos (FN): Es un tipo de error en el cual el modelo ha etiquetado un evento como "negativo" pero dicho evento es positivo.
- Falso Positivo (FP): Es un tipo de error en el cual el modelo ha etiquetado un evento como "positivo" pero dicho evento es negativo.

Estos cuatro valores son los que conforman lo que se conoce como matriz de confusión.

Para la evaluación se pueden calcular distintos medidas de efectividad del modelo como son:

$$\begin{cases} Precision = \frac{VP}{VP+FP} \\ NegativePredictiveValue = \frac{VN}{VN+FN} \\ Accuracy = \frac{VP+VN}{VP+VN+FP+FN} \\ Sensitivity = \frac{VP}{VP+FN} \\ Specificity = \frac{VN}{VN+FP} \end{cases} \tag{6}$$

La evaluación más simple es la que se mencionó en el anterior párrafo, se separa parte la data de entrenamiento en dos partes, una parte que se utilizará para entrenar el modelo, y otra parte que servirá para evaluar el modelo. De esta forma podemos hallar el valor de *Accuracy* de las fórmulas en la ecuación 6, el cual es un buen indicador del rendimiento del modelo. Esta forma de evaluación tiene algunas desventajas, por ejemplo, es posible que las muestras que se separen para hacer las pruebas, formen parte esencial para el entrenamiento del modelo y por tanto, quitar estas muestras de la data de entrenamiento puede ocasionar

un falso resultado de evaluación de que el modelo no es efectivo. Esta forma de evaluación sirve mejor cuando se tiene muchísimas muestras de entrenamiento.

Otra forma de evaluación es la que se determina con la técnica de "validación cruzada de K-iteraciones", al igual que la evaluación que se menciono en el anterior párrafo, esta técnica también separa una parte de la data para las pruebas, la diferencia consiste en que no se separa solo una parte arbitraria de la data sino que se separa la data entera en K diferentes divisiones de entrenamiento/evaluación, de modo que la 1/K parte de la data se usa para evaluación mientras lo demás se utiliza para entrenamiento, esto se realiza K-veces (K-iteraciones) tal que en cada iteración la data que se usa para evaluación nunca se repite, luego, al evaluar la data en cada iteración obtenemos K diferentes medidas de rendimiento (utilizando también la métrica de *Accuracy*), finalmente se promedian y ese es el valor de efectividad del modelo que se obtiene.

Este tipo de evaluación se suele utilizar cuando se tiene una cantidad limitada de datos de entrenamiento como es el caso de este trabajo. K generalmente se establece en un valor entre 5 y 10. El proceso general se ilustra en la figura 17, donde se muestra la división de la data para un ejemplo de 20 muestras con un K=4.

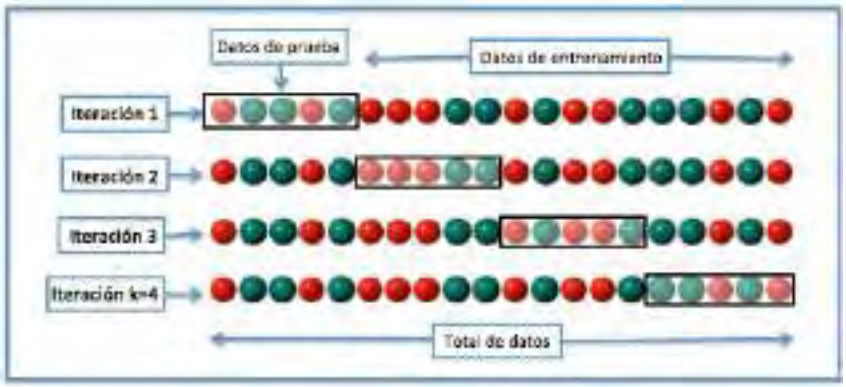

Figura 17: Validación cruzada de K iteraciones con K=4.

Fuente: Wikipedia

2.5. Clasificación Supervisada de Audio

Un clasificador de audio es un sistema o algoritmo encargado de clasificar eventos, es decir, un modelo encargado de diferenciar de forma autónoma un evento de otro. El desarrollo de este modelo o clasificador se divide en las siguientes etapas:

2.5.1. Recolección e Identificación de los Datos

La etapa de recolección e identificación consiste en la recolección de data cruda en el campo de acción y una organización de la data para discriminar la data relevante. En esta etapa se realiza la construcción de un set de datos que correspondan con los eventos que se quieren clasificar, por ejemplo, si se quiere entrenar un modelo que clasifique música de ruido, entonces se debe crear un set de datos con fragmentos o muestras de señales de audio, estas muestras deben tener sus respectivas etiquetas que indiquen si se trata de música o de ruido. En el etiquetado, un humano es el proporciona información relevante para el entrenamiento y es debido a esto que estos clasificadores son llamados como de tipo "supervisado".

En lo que respecta a la cantidad de datos o muestras que debe tener un set de datos, no es posible conocer exactamente la cantidad mínima ni máxima de datos que debe tener para que un entrenamiento sea efectivo. Aunque se recomienda comenzar con al menos 50 muestras (ver figura 20), en realidad depende de varios factores: el tipo de evento, la variedad de formas en que se puede presentar el evento, la calidad de las muestras del evento, etc. Si los eventos que se quieren detectar son de una gran variedad y se tiene muy pocas muestras de todas las variedades o solo muestras de algunas variedades entonces al entrenar nuestro modelo se puede generar lo que se conoce como "*Underfitting*", que quiere decir que nuestro algoritmo no fue capaz de aprender ningún patrón ya que no fue capaz de generalizar el conocimiento; por otro lado, aunque tengamos miles de muestras, si todas estas corresponden a variedades específicas, entonces el algoritmo solo habrá sido capaz de aprender a reconocer los eventos particulares con los que se le entreno y cuando el modelo entrenado bajo estas circunstancias intente clasificar una nueva variedad siempre fallará, esto es lo que se conoce como "*Overfitting*".

2.5.2. Pre-procesamiento de la Datos

El pre-procesamiento es una etapa preliminar a cualquier procesamiento o entrenamiento, el objetivo de esta etapa es transformar la data de modo que sea más fácil de utilizar en las posteriores etapas, por ejemplo en esta etapa se pueden realizar distintas técnicas como son: filtrado, limpieza de data, reducción de ruido, reducción de data, etc.

2.5.3. Extracción de Características y Definición del Training Dataset

Si bien después de haber realizado el pre-procesamiento ya se dispone de un set de datos con sus respectivas etiquetas, aún no se puede realizar un entrenamiento, esto es debido a que este set de datos esta compuesto de fragmentos de señales digitales que no son más que bits que no llevan mayor información relevante, por ello, para poder entrenar un clasificador se requiere extraer información relevante de estas muestras y organizarlas en un nuevo set de datos el cual llamaremos "*Training Dataset*". La información relevante que se extraerá de las muestras se denominan "*features*" o "características"

En procesamiento digital de señales, una característica consiste en un valor numérico o un conjunto de valores numéricos que pueden aportar cierta información de una señal, por ejemplo, el valor promedio (media aritmética) de los puntos de una señal puede aportar una información básica de cuál es la tendencia estadística central en la que se encuentran los valores de la señal.

Debemos recordar que, como se menciono en la sección 2.1.1, el infrasonido es por naturaleza una onda de sonido, por lo que su tratamiento y análisis es igual al de una onda sonora. Las características más usuales que se pueden extraer de las señales acústicas o sonoras figuran en la siguiente tabla.

Extracted Audio Features	
Zero Crossing Rate	Temporal features
Relative Difference Function	
Skewness	
Kurtosis	
Spectral Centroid	Spectral features
Spectral Spread	
Spectral Flux	
Spectral Slope	
Spectral Roll-off	
Spectral Skewness	
Spectral Kurtosis	
Inharmonicity	
Tristimulus I	
Odd to Even Harmonic Ratio	
Chroma (12)	
Spectral Flatness Measure (24)	
Spectral Crest Factor (24)	
Spectral Sharpness	
Spectral Smoothness	
Spectral Variability	
Irregularity	
Brightness	
Roughness	
MFCC (13)	Cepstral features
Delta-MFCC (13)	
BFCC (26)	
LPCC (12)	

Figura 18: Tabla con las características o *features* más usados en procesamiento de audio.
Fuente: https://www.researchgate.net/figure/List-of-extracted-features_tbl1_283348525

Como se observa, algunas de estas características corresponden al dominio del tiempo , otras al dominio de la frecuencia, y otras al dominio cepstral de una señal de audio. Aunque existen decenas de características, algunas son más relevantes que otras según el tipo de detección que se quiera realizar, el grupo de características más relevantes para cierta

aplicación es lo que llamamos un "patrón de características", este patrón es el que nos permitirá la detección del evento particular que nos interesa. Es una tarea importante del desarrollo de un clasificador la identificación de cuales serían las características que formarían un patrón que pueden ayudar a diferenciar los eventos que desean discriminar de otros.

El patrón de características a nivel práctico esta compuesto de vectores donde cada elemento del vector corresponde a una característica extraída de cada muestra de señal. De esta forma, el *Training Dataset* estará compuesto de vectores de características y las respectivas etiquetas de cada muestra.

En lo que respecta a clasificación de audio, los tipos de características mas utilizados son las de tipo espectral, a continuación se detallan algunos de las más utilizados en el entrenamiento de clasificadores para aprendizaje supervisado.

2.5.3.1. Centroide Espectral (Spectral Centroid)

El centroide espectral es una característica de tipo estadística que indica donde se localiza el "centro de masa" o "centro de gravedad" del espectro. Consiste en un único valor numérico y se halla considerando el espectro como una distribución de probabilidad, en donde se utiliza la amplitud en cada frecuencia para determinar la frecuencia media ponderada en amplitud. El centroide se calcula según la siguiente fórmula:

$$SC = \frac{\sum_{n=1}^{N} m(n) f(n)}{\sum_{n=1}^{N} m(n)} \tag{7}$$

Donde $f(n)$ es el valor de frecuencia correspondiente al bin n y $m(n)$ es el valor de los pesos en cada respectivo bin de frecuencia n (la magnitud del espectro en esa frecuencia).

2.5.3.2. Envergadura Espectral (Spectral Spread)

La envergadura espectral mide la forma en como esta distribuido el espectro alrededor del centroide, bajos valores de esta medida corresponden a señales cuyo espectro se encuentra muy concentrado alrededor del centroide y de forma inversa para valores altos. Esta medida consiste de un solo valor y para poder calcularla, tenemos que tomar la varianza del espectro del centroide espectral de acuerdo con la siguiente fórmula:

$$SSp = \frac{\sum_{n=1}^{N} m(n)(f(n) - SC)^2}{\sum_{n=1}^{N} m(n)} \tag{8}$$

Donde $f(n)$ es el valor de frecuencia correspondiente al bin n, $m(n)$ es el valor de los pesos en cada respectivo bin de frecuencia n (la magnitud del espectro en esa frecuencia) y $(f(n) - SC)$ denota la distancia de la banda de frecuencia hasta el centroide (SC).

La raíz de la envergadura espectral es la desviación estándar del espectro del centroide, este valor será útil para hallar las características de Asimetría y Kurtosis Espectral.

2.5.3.3. Asimetría Espectral (Spectral Skewness)

La asimetría espectral también mide la forma en como esta distribuido el espectro alrededor del centroide, el valor de esta característica es cero si la distribución es completamente simétrica alrededor del centroide, si la distribución es asimétrica hacia la derecha toma valores negativos y si la distribución es asimétrica hacia la izquierda toma valores positivos. Esta medida consiste de un solo valor y para poder calcularla, usaremos los valores del centroide y de la envergadura espectral de acuerdo con la siguiente fórmula:

$$SSk = \frac{\sum_{n=1}^{N} m(n)(f(n) - SC)^3}{\sum_{n=1}^{N} m(n)} \left(\frac{1}{\sqrt{SSp}^3}\right) \tag{9}$$

Donde $f(n)$ es el valor de frecuencia correspondiente al bin n, $m(n)$ es el valor de los pesos en cada respectivo bin de frecuencia n (la magnitud del espectro en esa frecuencia), $(f(n) - SC)$ denota la distancia de la banda de frecuencia hasta el centroide (SC) y SSp es el valor de la envergadura espectral.

2.5.3.4. Kurtosis Espectral (Spectral Kurtosis)

La kurtosis espectral es otra caracerística que mide la forma en como esta distribuido el espectro alrededor del centroide, esta mide el grado de “planicie” del espectro alrededor del centroide, viendolo de otra forma, también se usa para medir la “picudez"del espectro, por ejemplo, un valor bajo de kurtosis indica un espectro uniformemente distribuido tal como el del ruido blanco. El valor de esta característica es 3 cuando el espectro tiene la forma de una distribución normal o gaussiana, valores menores a 3 indican una distribución más plana y mayores a 3 una distribución más picuda. Esta medida consiste de un solo valor y para poder calcularla, usaremos los valores del centroide y de la envergadura espectral de acuerdo con la siguiente fórmula:

$$SK = \frac{\sum_{n=1}^{N} m(n)(f(n) - SC)^4}{\sum_{n=1}^{N} m(n)} \left(\frac{1}{\sqrt{SSp}^4}\right) \tag{10}$$

Donde $f(n)$ es el valor de frecuencia correspondiente al bin n, $m(n)$ es el valor de los pesos en cada respectivo bin de frecuencia n (la magnitud del espectro en esa frecuencia), $(f(n) - SC)$ denota la distancia de la banda de frecuencia hasta el centroide (SC) y SSp es el valor de la envergadura espectral.

2.5.3.5. Roll-off Espectral (Spectral Roll-off)

El roll-off espectral es una característica que mide el valor de frecuencia bajo el cual esta concentrado un cierto porcentaje (para este trabajo es 85 %) de energía con respecto a la distribución total del espectro. Esta medida consiste de un solo valor y se calcula de acuerdo con la siguiente fórmula:

$$PuntoRollOff = n_{RO}, \;\; tal\ que: \;\; \sum_{n=1}^{n_{RO}} (f(n))^2 = 0.85 \sum_{n=1}^{N} (f(n))^2 \tag{11}$$

Donde $f(n)$ es el valor de frecuencia correspondiente al bin n. El punto de Roll-off (n_{RO}) mostrado en la fórmula calcula el bin de frecuencia donde se encuentra, pero si se quiere hallar la frecuencia de Roll-off, el bin puede ser transformado a su valor en Hz multiplicándolo por $f_s/2N$, donde fs es la frecuencia de muestreo de la señal.

2.5.3.6. Pendiente Espectral (Spectral Slope)

La característica de pendiente espectral mide el nivel de decaimiento de la magnitud del espectro en la banda de baja frecuencia respecto a la de alta frecuencia, su valor se obtiene calculando una regresión lineal. Esta medida consiste de un solo valor y se calcula de acuerdo con la siguiente fórmula:

$$SSl = \frac{1}{\sum_{n=1}^{N} m(n)} \frac{N\sum_{n=1}^{N} f(n)m(n) - \sum_{n=1}^{N} f(n) \sum_{n=1}^{N} m(n)}{N\sum_{n=1}^{N} (f(n))^2 - (\sum_{n=1}^{N} f(n))^2} \tag{12}$$

Donde $f(n)$ es el valor de frecuencia correspondiente al bin n y $m(n)$ es el valor de los pesos en cada respectivo bin de frecuencia n (la magnitud del espectro en esa frecuencia).

2.5.3.7. Puntos de Giro Positivos Espectrales (Spectral Positive Turning Points) Esta característica mide la cantidad de puntos de giro positivos de la distribución del espectro de una señal, un punto de giro es el punto donde una señal cambia de un tipo de concavidad a otra. En señales discretas, un punto de giro positivo se da cuando la resta de la magnitud de dicho punto con un punto anterior es un valor negativo. Para propósitos prácticos, los puntos

de giro positivos representan los puntos máximos locales (picos) de una señal. Esta medida consiste de un solo valor y se calcula de acuerdo con la siguiente fórmula:

$$\begin{cases} SPTP = \sum_{n=1}^{N-1} |signo[m_{diff}(n)] - signo[m_{diff}(n-1)]| \\ m_{diff}(n) = m(n+1) - m(n) \end{cases} \tag{13}$$

Donde $m(n)$ es el valor de la magnitud en cada respectivo bin de frecuencia n (la magnitud del espectro en esa frecuencia) y m_{diff} es una función que esta compuesta de la diferencia discreta de los valores consecutivos de $m(n)$; además la función signo esta dada por la siguiente ecuación:

$$signo[m_{diff}(n)] = \begin{cases} 1, & \text{si } m_{diff}(n) \geq 0, \\ -1, & \text{si } m_{diff}(n) < 0. \end{cases} \tag{14}$$

2.5.3.8. Relación de Energía entre Bandas (Band Energy Rate)

Esta característica mide la relación de energía contenida en dos bandas del espectro de una señal, a diferencia de las demás características, se puede obtener muchos valores, según las bandas de las cuales queramos encontrar su relación. La fórmula para hallar esta característica se calcula de acuerdo con la siguiente fórmula:

$$BER = \frac{\sum_{n=a_1}^{a_2} (m(n))^2}{\sum_{n=b_1}^{b_2} (m(n))^2} \tag{15}$$

Donde $m(n)$ es el valor de la magnitud en cada respectivo bin de frecuencia n (la magnitud del espectro en esa frecuencia), a_1 y a_2 son los bins de frecuencia que corresponden a los límites de la primera banda que entrará a la relación, b_1 y b_2 son los bins de frecuencia que corresponden a los límites de la segunda banda. Por ejemplo, en la figura 19 se desea hallar la relación entre dos bandas para cierta ventana, se ha marcado las áreas de energía que corresponden a la energía concentrada en la baja frecuencia (área verde) entre la energía concentrada en alta frecuencia (área azul), para este ejemplo en particular, el límite inferior de la banda de baja frecuencia sería el primer bin ($a_1 = 1$) y el límite superior de la banda de alta frecuencia sería el ultimo bin ($b_2 = N$), y además, ambos dividen exactamente en dos el espectro total por lo que los límites restantes serían iguales ($a_2 = b_1$), solo quedaría definir en exactamente que bin se desea dividir el espectro para de esta forma tener el valor para

todos los límites y poder calcular el valor de la relación de energía entre bandas según la ecuación 15.

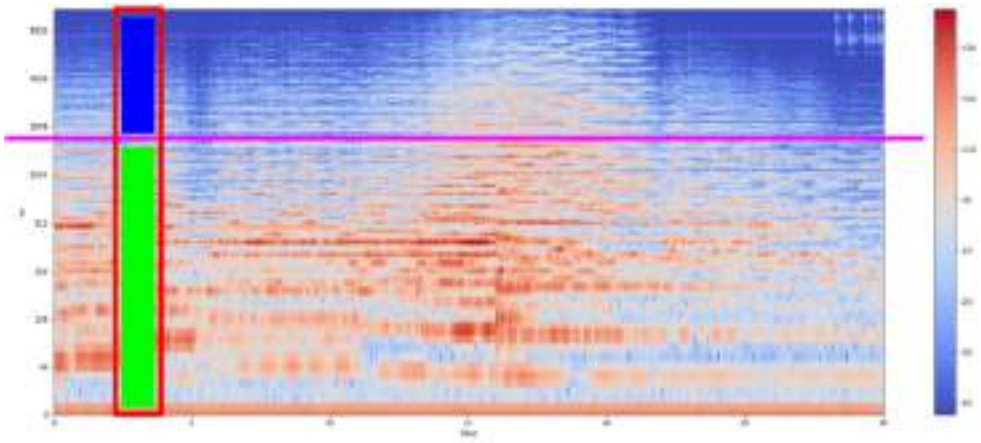

Figura 19: Representación de la energía de dos bandas en una ventana de cierta señal, en verde la banda de baja frecuencia y en azul la banda de alta frecuencia.

Fuente: https://github.com/musikalkemist/AudioSignalProcessingForML/

2.5.3.9. Tasa de Cruce por cero (Zero Crossing Rate)

A diferencia de las demás características, la tasa de cruce por cero es una característica temporal que nos indica cuantas veces la señal cruza el nivel cero en el dominio del tiempo, en otras palabras, es la cantidad de veces que la señal cruza de positivo a negativo y viceversa. Esta medida consiste de un solo valor y se calcula de acuerdo con la siguiente fórmula:

$$ZCR = (\sum_{i=1}^{I-1} |signo[x(i)] - signo[x(i-1)]|)\frac{fs}{2I} \tag{16}$$

Donde x es la amplitud de la señal en tiempo para cada muestra i, I es la cantidad total de muestras de la señal y f_s es la tasa de muestreo; además la función signo esta dada de la misma forma que la ecuación 14, con la diferencia de que aquí aplica $x(i)$ en lugar de $m_{diff}(n)$.

2.5.4. Algoritmo de Machine Learning y Entrenamiento

La elección de este algoritmo es muy importante; este se elige según la tarea que se quiere realizar, en la documentación de la librería *Scikit-learn* se encuentra un esquema muy útil y didáctico para elegir correctamente el algoritmo (ver figura20).

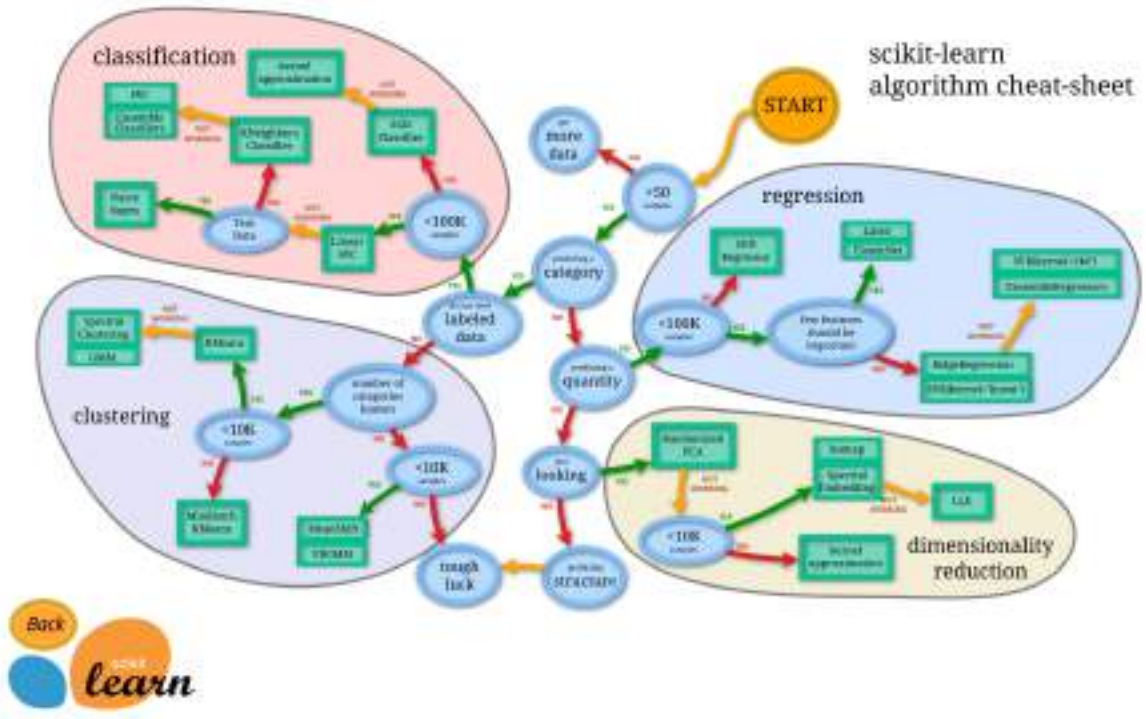

Figura 20: Esquema de ayuda para la elección de un algoritmo óptimo de *Machine Learning*

Fuente: Documentación de la Librería Python "Scikit-learn"

El entrenamiento de un modelo requiere de la elección de un algoritmo de *Machine Learning*, dos de los algoritmos más utilizados son los algoritmos de: K vecinos más cercanos (*K-Nearest-Neighbor*) y Maquina de Vectores de Soporte (*Support Vector Machine* ó *Support Vector Classifier*), también se puede observar en el esquema de la figura 20, que estos dos algoritmos son los apropiados para trabajar en tareas de clasificación. Cabe mencionar que cada algoritmo tiene parámetros que pueden ser modificados para manipular el proceso de aprendizaje, estos parámetros son también llamados hiperparámetros y dependiendo de los valores que se elija, el modelo puede tener mejores o peores resultados. A continuación se presenta un breve detalle de ambos algoritmos:

2.5.4.1. K vecinos más cercanos (K-Nearest-Neighbor)

El algoritmo KNN (*K-Nearest-Neighbor*) es un tipo de algoritmo de aprendizaje supervisado usado en regresión y clasificación. El algoritmo KNN sirve para realizar predicciones de clases, para ello primero calcula la distancia entre el punto a predecir y los puntos de la data de entrenamiento, luego selecciona los K puntos más cercanos al punto a

predecir y calcula la clase con mayor probabilidad, osea, la clase con mayor número de muestras entre los k vecinos, para finalmente asignarle esta clase al punto a predecir. Para este trabajo, los "puntos" a predecir y los "puntos" de entrenamiento vendrían siendo los vectores de características, es decir, estos puntos vendrían siendo vectores en un espacio de *n*-dimensiones (siendo *n* la longitud del vector de características).

La elección del valor de K, es muy importante y puede ser crucial para la correcta predicción de una determinada muestra, por ejemplo, en la figura 21 se observa los puntos azul y verde que representan a la data de entrenamiento, allí se desea predecir la clase de la estrella roja, si se elige K=3 se le asignaría una clase verde pero si se elige K=6 se le asignaría una clase azul.

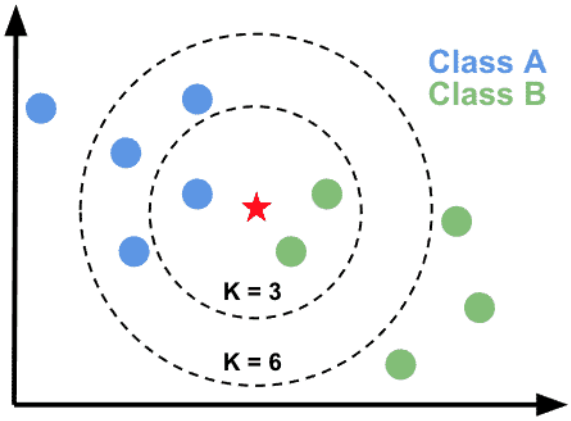

Figura 21: Espacio donde se muestran los puntos de entrenamiento correspondientes a dos clases (azul y verde) y el punto, cuya clase se quiere predecir, representado por una estrella siendo evaluada para un K=3 y un K=6.

Fuente: https://www.jcchouinard.com/k-nearest-neighbors/

A parte del valor de K, también se debe elegir si en la predicción de la clase, todos los vecinos dentro de los K-vecinos tienen un voto uniforme, o si los que se encuentren más cercanos tienen un voto con "más peso", este hiperparámetro se le conoce como el valor de "weigths" y puede tomar los dos valores explicados anteriormente: "uniform" o "distance".

Otro hiperparámetro importante es el método que se utilizará para hallar la distancia, existen varios métodos, la forma más simple de hallar la distancia, es utilizar la distancia Euclídea, que es una línea recta calculada entre dos puntos en un plano n-dimensional. La distancia de Manhattan, también llamada, distancia de cuadra de la ciudad, ya que se

visualiza como la distancia entre dos puntos que pasa por las líneas de una cuadrícula y no en una línea recta directa (como si fueran las calles de una ciudad). La distancia Minkowski, es la forma generalizada para las distancias Euclídea ($r = 2$) y Manhattan ($r = 1$).

$$\begin{cases} D.Euclidea : D(x,y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \\ D.Manhattan : D(x,y) = \sum_{i=1}^{n} |x_i - y_i| \\ D.Minkowski : D(x,y) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r} \end{cases} \tag{17}$$

Donde x y y son los dos puntos cuya distancia se quiere encontrar, y n es la dimensionalidad de los puntos.

2.5.4.2. Maquina de Vectores de Soporte (Support Vector Machine)

El algoritmo de *Support Vector Machine* (SVM) es un algoritmo de aprendizaje supervisado cuyo objetivo es dividir dos conjuntos de puntos de distintas clases mediante un hiperplano, el algoritmo de SVM trata de encontrar la forma más óptima de realizar esta separación, de forma que el plano se encuentre lo más separado posible de ambas clases, este margen de separación esta definido por lo que se conoce como los vectores de soporte. En la figura 22 se observa el hiperplano hallado para los puntos de clases azul y rojo, en un plano bidimensional el hiperplano es una línea, notar que, si simplemente quisiéramos separar las dos clases, la línea podría ser graficada de varias formas, lo que hace el algoritmo de SVM es encontrar la posición en la que esta línea tenga el margen máximo de separación entre ambas clases.

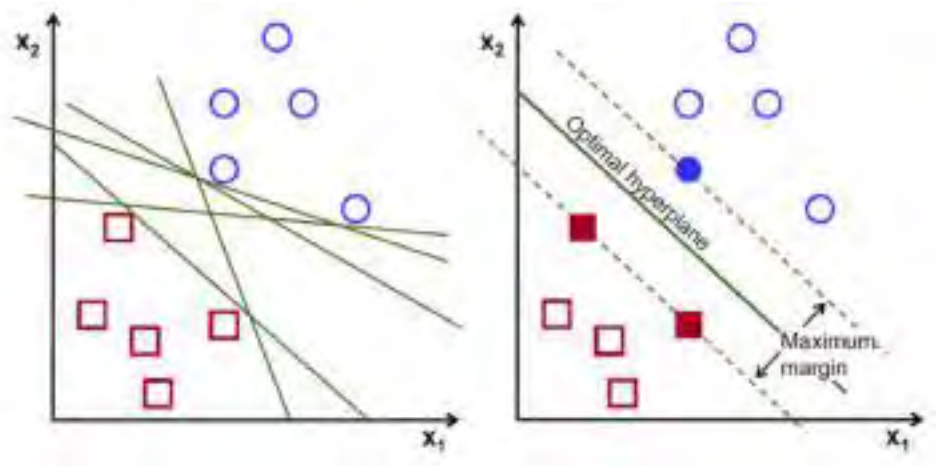

Figura 22: En la primera imagen se ven los posibles hiperplanos para separar las clases, en la segunda imagen, el hiperplano óptimo. Los puntos remarcados son los vectores de soporte.

Fuente: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Ahora se hará una revisión de la teoría básica del algoritmo de SVM. Supongamos que tenemos un conjunto de puntos de entrenamiento etiquetados tal como en la figura 22, de modo que cada punto esta definido por las dos variables: $(x_1, y_1), ..., (x_i, y_i)$, tal que x_i es el punto que puede ser de N-dimensiones ($x_i \in \Re^N$) y y_i es su etiqueta o clase; puesto que el algoritmo SVM esta especializado para clasificaciones binarias, entonces de forma general y_i debe ser +1 para clases positivas y -1 para clases negativas. De esta forma, el hiperplano óptimo debe ser una función que pueda predecir el y_i para un x_i desconocido, si la data de entrenamiento es linealmente separable, dicho hiperplano debe estar definido de la siguiente forma:

$$\begin{cases} w^T x_i + b \geq 1 & \text{para todo } x_i \in clasepositiva \\ w^T x_i + b \leq -1 & \text{para todo } x_i \in clasenegativa \end{cases} \tag{18}$$

Donde x representa la variable de entrada, w es un vector que representa pesos y b una simple constante. De lo ya visto, la regla de decisión queda de la forma siguiente:

$$f(x) = signo[w.x + b] \tag{19}$$

La optimización se puede resolver introduciendo un Lagrangiano de la siguiente forma:

$$L(w,b,\alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i((x_i.w)+b) - 1 \qquad \alpha = (\alpha_i, ..., \alpha N) \tag{20}$$

Donde $\alpha_i \geq 0$ son los multiplicadores de Lagrange que forman el vector α y N es el número total de muestras en el Dataset de entrenamiento. Si se aplica el teorema de Kuhn-Tucker, que declara que las derivadas parciales del lagrangiano L con respecto a w y b son iguales a cero. Con este teorema se obtiene las siguientes condiciones:

$$\begin{cases} \sum_{i=1}^{N} \alpha_i y_i = 0 \\ w = \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \end{cases} \tag{21}$$

Según el teorema, α_i debe satisfacer las condiciones de Karush-Kuhn-Tucker, de modo que las muestras del *Training Dataset* que no entren en el margen óptimo deben tener un α_i nulo, siendo los demás, los vectores de soporte. Según lo explicado, la regla de decisión podría quedar de la siguiente manera:

$$f(x) = signo[\sum_{i=1}^{N} \alpha_i y_i (x_i.x_j) + b] \tag{22}$$

Donde x_i son los puntos de entrenamiento, x_j es el nuevo punto que se desea clasificar.

Finalmente, aunque solo se hará una breve introducción, es importante conocer el concepto de Kernel para el algoritmo de SVM. Supongamos que nuestra data tiene la forma que se muestra en la primera gráfica de la figura 23, en aquí se puede observar que las clases están claramente separadas y la clasificación no debería ser díficil, sin embargo el hiperplano óptimo no podría ser encontrado, debido a que es imposible dibujar una línea recta que pueda dividir la data en dos secciones, para estos casos se usa el llamado “truco del kernel”, que transforma el espacio donde se encuentran los puntos, de modo que, ahora es posible añadir un nueva dimensión que haga posible la separación de las clases con un hiperplano, este hiperplano en un plano transformado se puede observar en la segunda imagen de la figura 23. La proyección resultante del hiperplano en el plano bidimensional se puede observar en la tercera imagen.

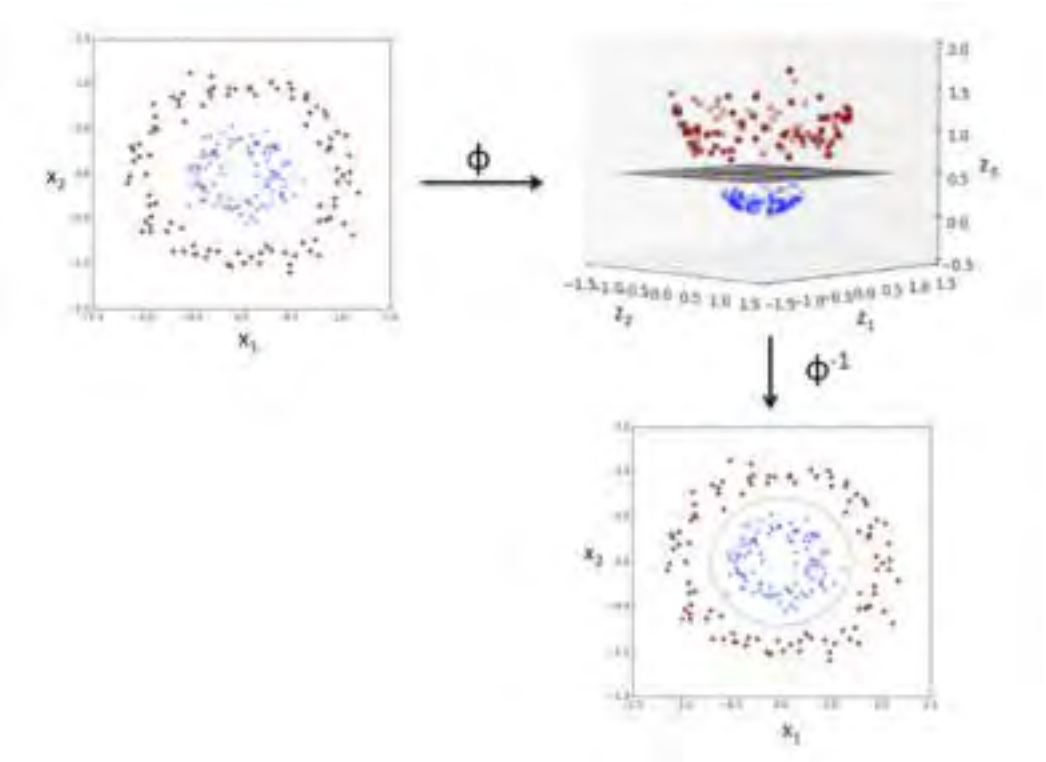

Figura 23: Transformación del espacio donde se encuentran los puntos de entrenamiento mediante el “truco del Kernel”.

Fuente: https://sebastianraschka.com/faq/docs/select_svm_kernels.html

Los distintos tipos de kernel que usualmente se utilizan se pueden observar en la imagen 24, se muestran el kernel lineal, el kernel de base radial (RBF) y el kernel polinómico. Para el presente proyecto, se trabajará con el kernel RBF, ya que es el kernel más utilizado y el más óptimo para trabajar con tipos de data no lineales.

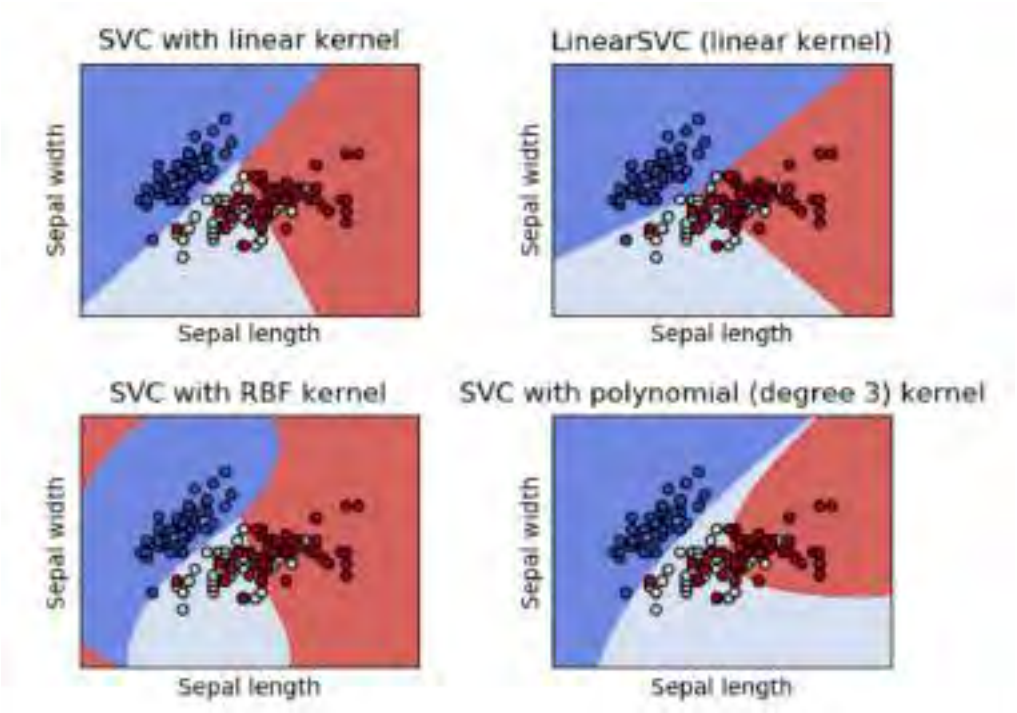

Figura 24: Aplicación de distintos Kernels en una data de entrenamiento compuesta por 3 clases.

Fuente: https://scikit-learn.org/stable/modules/svm.html

Los hiperparámetros para el algoritmo SVM están ligados al Kernel, ya que para este trabajo se utilizará el kernel RBF, entonces los hiperparámetros que se deben ajustar son dos: C y *gamma*. El parámetro *gamma* define cuán lejos debe llegar la influencia de un solo punto de la data de entrenamiento, bajos valores de *gamma* se corresponden a una influencia más lejana, y valores altos a una influencia más cercana. El parámetro C define cuánto tiene permitido el algoritmo dejar pasar clasificaciones erróneas, mientras más alto sea el valor C, menos clasificaciones erróneas se permitirán a la hora de calcular el hiperplano que separe las clases, lo usual es que el valor de C tenga un valor medio, ya que un valor más bajo puede permitir una mejor generalización de las clases con el costo de un número pequeño de clasificaciones erróneas. En la imagen 25 se puede observar que, aunque aparentemente los modelos con valor altísimo de C generan la separación más perfecta, quizás el modelo más óptimo sea uno que permita algunas clasificaciones erróneas (como el observado con $C = 1$ y $gamma = 0.1$), esto debido a que ese par de clasificaciones erróneas pueden ser simplemente valores atípicos que no representan a sus respectivas clases y no deberían influenciar demasiado en el posicionamiento del hiperplano.

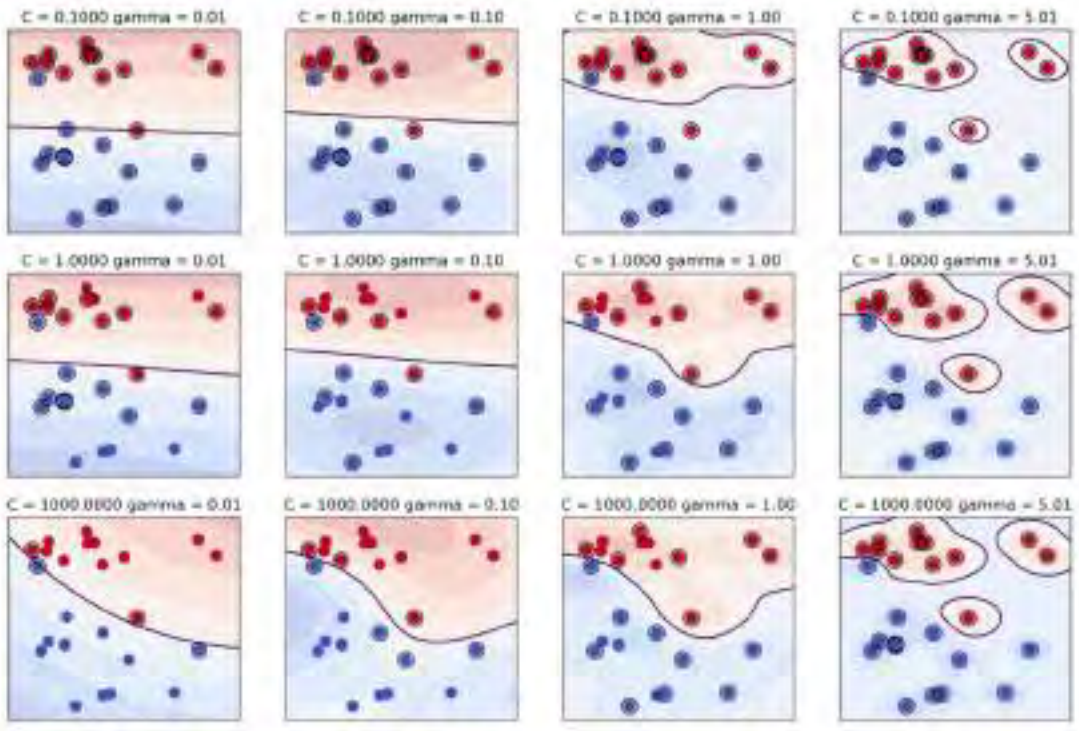

Figura 25: Separación de dos clases bajo distintos valores de *C* y *gamma*.

Fuente:

https://amueller.github.io/aml/02-supervised-learning/07-support-vector-machines.html

La elección de los valores de los hiperparámetros *C* y *gamma* pueden influenciar en gran medida el rendimiento del modelo, por ello es muy importante elegir los valores más adecuados.

2.5.5. Evaluación del Clasificador

La etapa final corresponde a una etapa de evaluación, la cual se puede realizar hallando los valores de precisión, *accuracy* o utilizando la técnica de Validación Cruzada explicada en la sección 2.4 . Según los resultado obtenidos en la evaluación, se aceptará o no el clasificador, en caso de no ser aceptado, se puede recurrir a recolectar más data, a utilizar otros métodos de limpieza de datos, probar con más características o ajustar los parámetros de los algoritmos en el entrenamiento, esto debe realizarse hasta obtener el clasificador/modelo más óptimo.

De manera general, el esquema completo del proceso de desarrollo de un clasificador

supervisado de audio se muestra en la figura 26.

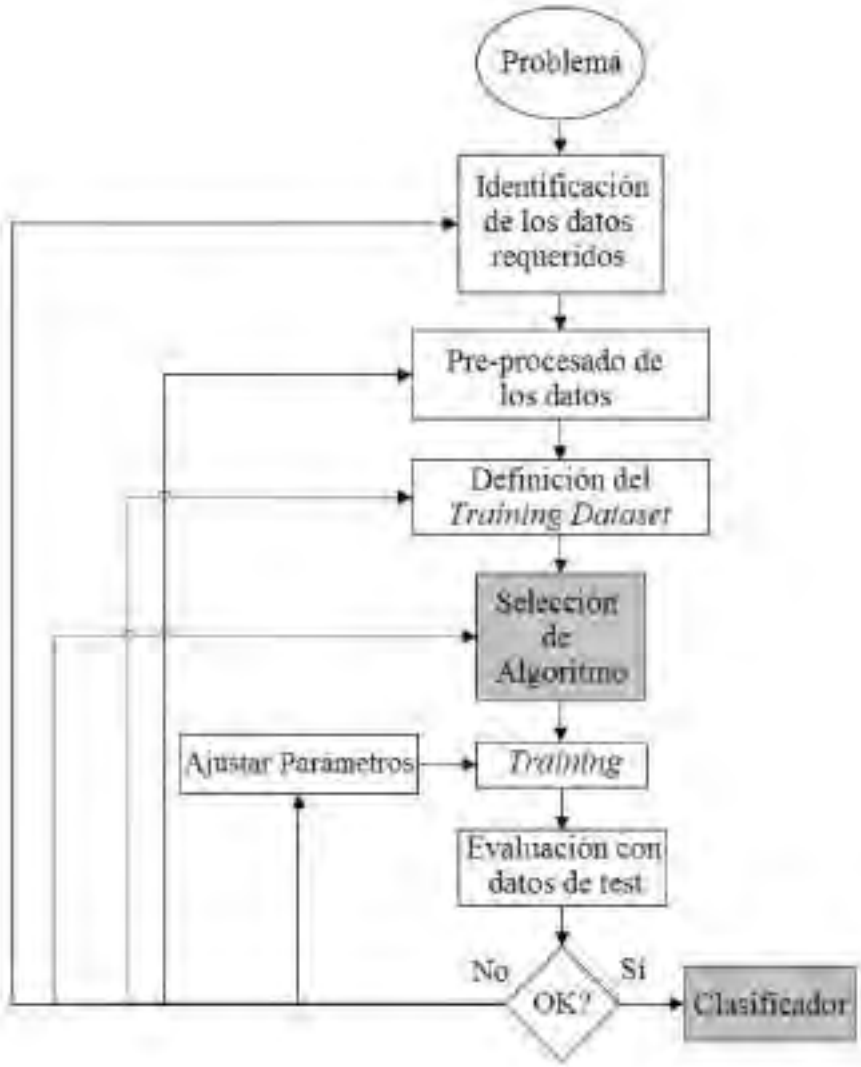

Figura 26: Proceso de desarrollo de un clasificador supervisado.

Fuente: [34]

III. Diseño

3.1. Estado Actual del Sistema

Es relevante mencionar que actualmente ya existen sistemas de monitoreo en la Laguna Palcacocha, estos no son sistemas de detección sino que simplemente cumplen con una función de observación de los eventos que acontecen en la laguna. El sistema de detección de avalanchas planteado en este trabajo de investigación tiene la función de complementar los sistemas de monitoreo existentes. Dichos sistemas de monitoreo se detallan a continuación:

3.1.1. Cámara de Videovigilancia en la laguna Palcacocha

Este sistema, implementado por el Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña (INAIGEM) en el 2017, tiene como objetivo monitorear cualquier eventualidad y advertir de los peligros asociados a la laguna Palcacocha o los glaciares en sus cercanías, incluyendo las avalanchas.

El sistema consiste de una cámara instalada en una de las morrenas cercanas a la laguna Palcacocha, el sistema esta conectado a la red de internet por medio de un radioenlace que la conecta con la sede del INAIGEM en la ciudad de Huaraz. El campo de visión de la cámara cubre la laguna y las paredes frontales de los glaciares Pucaranra y Pallcaraju, y Las imágenes captadas por la cámara son transmitidas en tiempo real a través de *Youtube* de esta manera, la población puede acceder a visibilizar la dinámica de la laguna.

Figura 27: Izquierda: Sistema de monitoreo ubicado en la laguna Palcacocha. Derecha: Imágenes obtenidas por la cámara en tiempo real.

Fuente: INAIGEM

3.1.2. Estación Meteorológica

En colaboración entre el INAIGEM, el Centro de Investigación Ambiental para el Desarrollo (CIAD) de la Universidad Nacional Santiago Antúnez de Mayolo (UNASAM) y la Gerencia de Recursos Naturales del Gobierno Regional de Áncash; fue instalada en la laguna Palcacocha una estación meteorológica portátil para el monitoreo y reporte de distintas variables climáticas: precipitación, humedad, viento, radiación y temperatura. Funciona durante todo el día, los 365 días del año, alimentado por paneles solares, la información recolectada por la estación es útil para conocer la evolución del clima en la laguna y sus cercanías.

Figura 28: Estación meteorológica portátil en la laguna Palcacocha.

Fuente: INAIGEM

3.2. Consideraciones Previas del Diseño

3.2.1. Elección del Sensor de Infrasonido

Para la elección del sensor se hizo una comparación entre los sensores de infrasonido más baratos que existen en el mercado y de los cuáles se tiene disponibilidad de compra.

En la tabla 2 se observa la comparación de especificaciones técnicas, requerimientos especiales y costos de los distintos sensores. A nivel técnico, algunos sensores son mejores que otros, por ejemplo los sensores de la marca Chaparral inducen menos ruido en las mediciones y su rango de funcionamiento en frecuencia es más amplio (ver tabla 2), sin embargo, su instalación e implementación requiere de sistemas más complejos y de tener otros equipos ya adquiridos (como un *Data Logger*) debido a que es un sensor analógico, además sus precios son los más altos entre todos los sensores comparados. Por otro lado, los sensores *Infiltec* y *Raspberry Boom* son ideales para este trabajo porque ambos ya disponen de un convertidor analógico digital, por lo tanto, se encuentran listos para adquirir, procesar y almacenar las señales, en particular, el sensor *Raspberry Boom* cuenta con la ventaja de que trabaja con el ordenador de placa reducida *Raspberry Pi*, haciendo que la instalación del sistema de adquisición de señales pueda ser más portable y compacta.

Por las razones anteriormente expuestas, para este trabajo se decidió adquirir el sensor de infrasonido *Raspberry Boom* de la marca *Raspberry Shake*.

PRODUCTO	ESPECIFICACIONES	DETALLES ADICIONALES Y REQUERIMIENTOS ESPECIALES	COSTO
Sensor Chaparral Model 64S	• Salida tipo: Diferencial • Ruido auto-inducido: Menos de 2 mPa RMS, 0.05 a 200 Hz Menos de 0.39 mPa RMS, 0.5 a 2 Hz • Alimentación: 12 v (11.25-20 v) DC. • Uso de corriente: Menos de 12 mA a 12.6 v •Sensibilidad nominal: 0.4v / Pascal • Rango de trabajo (Frecuencia): Rango inaudible (0.02 Hz – 20Hz) Rango audible (20 – 245 Hz) • Dimensiones: altura max. (7.87 cm) y diametro max. (9.25 cm). • Peso: 0.57 Kg	Sensor de tipo analógico, la salida es entregada en voltios. Requiere de Data Logger. El sensor tiene una salida de tipo DBPE A054-130 y el conector del cable a utilizar debe ser de tipo S 102 A054-130+. Este modelo tiene 3 ports (entradas acústicas).	$4200
Sensor Chaparral Model 60S	• Salida tipo: Diferencial • Ruido auto-inducido: Menos de -62dB Pa²/Hz, rel a 1 Pa Menos de 3 mPa RMS, 0.1 a 40 Hz Menos de 0.8 mPa RMS, 0.5 a 2 Hz • Alimentación: 12 v (11.25-20 v) DC. • Uso de corriente: Menos de 12 mA a 12.6 v •Sensibilidad nominal: 0.4v / Pascal • Rango de trabajo (Frecuencia): Rango inaudible (0.03 Hz – 20Hz) Rango audible (20 – 245 Hz) • Dimensiones: altura max. (4.3 cm) y diametro max.(9.5 cm). • Peso: 0.2 Kg	Sensor de tipo analógico, la salida es entregada en voltios. Requiere de Data Logger. El sensor tiene una salida de tipo DBPE A054-130 y el conector del cable a utilizar debe ser de tipo S 102 A054-130+. Este modelo solo tiene un port.	$4200
Sensor Chaparral Model 21 o 25	• Salida tipo: Diferencial • Ruido auto-inducido: Menos de -62dB Pa2/Hz, rel a 1 Pa Menos de 3 mPa RMS, 0.1 a 40 Hz Menos de 0.8 mPa RMS, 0.5 a 2 Hz • Alimentación: 12 v (9-18 v) DC • Uso de corriente: Menos de 40 mA a 12 v •Sensibilidad nominal: 0.4v a 2v / Pascal • Rango de trabajo (Frecuencia): Rango inaudible (0.1 Hz – 20Hz) Rango audible (20 – 200 Hz) • Dimensiones: altura max. (18 cm) y diametro max. (18 cm) (Modelo 21). altura max. (14 cm) y diametro max. (23 cm) (Modelo 25). • Peso: 1.5 Kg (Modelo 21). 2.4 Kg para la version de 4 ports (Modelo 24).	Sensor de tipo analógico, la salida es entregada en voltios. Requiere de Data Logger. El sensor tiene una salida de tipo PT07E-12-8P/Bulgin Buccaneer 400 series (puede elegirse cualquiera de los dos al momento de comprar) y el conector del cable a utilizar debe ser de tipo PT06E-12-8S-SR con el primer tipo de salida y PX0410/08S/6065 con el segundo tipo de salida. El modelo 21 solo tiene un port.	$4200
Sensor Chaparral Model 20 o 24	• Salida tipo: Diferencial • Ruido auto-inducido: Menos de -62dB Pa2/Hz, rel a 1 Pa Menos de 3 mPa RMS, 0.1 a 40 Hz Menos de 0.8 mPa RMS, 0.5 a 2 Hz • Alimentación: 12 v (9-18 v) DC. • Uso de corriente: Menos de 40 mA a 12 v •Sensibilidad nominal: 0.4v / Pascal • Rango de trabajo (Frecuencia): Rango inaudible (0.1 Hz – 20Hz) Rango audible (20 – 200 Hz) • Dimensiones: altura max. (18 cm) y diametro max. (18 cm) (Modelo 20). altura max. (14 cm) y diametro max. (23 cm) (Modelo 24). • Peso: 1.5 Kg (Modelo 20). 2.4 Kg para la version de 4 ports (Modelo 24).	Sensor de tipo analógico, la salida es entregada en voltios. Requiere de Data Logger. El sensor tiene una salida de tipo PT07E-12-8P/Bulgin Buccaneer 400 series (puede elegirse cualquiera de los dos al momento de comprar) y el conector del cable a utilizar debe ser de tipo PT06E-12-8S-SR con el primer tipo de salida y PX0410/08S/6065 con el segundo tipo de salida. El modelo 20 solo tiene un port.	$3500
Sensor Infiltec Model INFRA20	• Muestras por segundo: 50 (50 Hz) • Sensibilidad: 1,000 counts/Pascal +/- 10 % precisión • Digitalizador: 16-bits • Rango de trabajo (Frecuencia): Rango inaudible (0.05 Hz – 20Hz) •Espacio usado almacenamiento: 10 Mb / día •Ruido auto-inducido: menor a 30 mPa o 63.5 dB SPL, 0.1 a 20 Hz	Sensor viene con digitalizador y funciona con software de uso libre para el procesamiento de data, lo que ahorra el uso de Data Logging. La data de este sensor se obtiene a través de un ordenador y su alimentación tambien se hace a traves de este (por medio de un puerto USB).	$345
Sensor Raspberry BOOM	• Muestras por segundo: 100 (100 Hz) • Sensibilidad: 56,000 counts/Pascal +/- 10 % precisión • Digitalizador: 24-bit • Rango de trabajo (Frecuencia): Rango inaudible (0.05 Hz – 20Hz) Rango audible (20 – 40 Hz) •Espacio usado almacenamiento: 15 Mb / día •Ruido auto-inducido: -35 dB, 1 a 50 Hz	Sensor viene con digitalizador y funciona con software de uso libre para el procesamiento de data, lo que ahorra el uso de Data Logging. La data de este sensor se obtiene a través de un ordenador de placa reducida (Raspberry Pi) y su alimentación tambien se hace a traves de este.	$375

Tabla 2: Comparación de distintos sensores de infrasonido

3.2.2. Formato de la Data de la Señal

El sensor *Raspberry Boom* entrega la data en un formato que se llama *miniSEED*, este tipo de data proviene de un formato llamado SEED (Standard for the Exchange of Earthquake Data), este formato es un estándar de intercambio de data sobre movimientos sísmicos y metadata relacionada; puntualizando, el formato *miniSEED* se refiere a un subconjunto del formato SEED que es usado para el manejo de data de señales en función del tiempo. Una de las principales características del formato *miniSEED* es que además de que contiene la data de las señales sin ninguna comprensión y su data de tiempo, contiene muy poca metadata, limitándose a solo lo más esencial como la tasa de muestreo o simples alarmas cuando suceda algún cambio en la tasa de muestreo de la señal entre otros problemas relacionados a la integridad de la señal.

Aunque en el párrafo anterior se mencionó que el formato *miniSEED* es utilizado para manejar data sobre movimientos sísmicos, vale indicar que también es bastante utilizado para el manejo de data de infrasonido, esto es así, ya que muchas veces ambas señales están relacionadas, y existen muchos proyectos e investigaciones donde se complementan sensores sísmicos y de infrasonido por lo que es muy útil que ambos compartan el mismo formato.

Es importante mencionar también que, debido a que este tipo de formato es en general muy poco utilizado fuera del ámbito académico, existen pocas herramientas para su análisis y tratamiento. Para el análisis y visualización utilizaremos el software de código abierto SWARM, un software creado y mantenido por el Servicio Geológico de Estados Unidos (*United States Geological Survey* o USGS), este software nos permitirá visualizar la data tal como se observa en la figura 29.

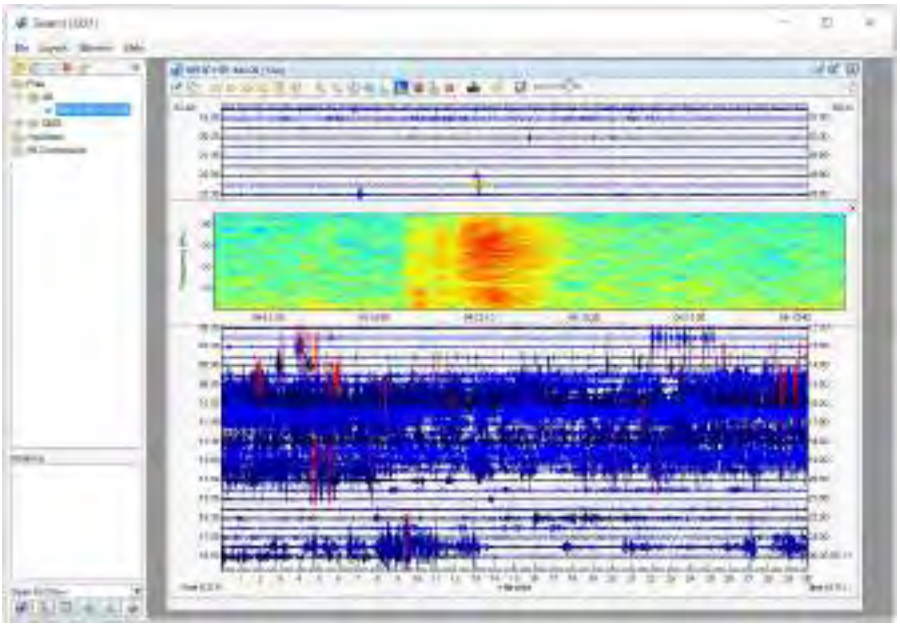

Figura 29: Muestra de la interfaz del software SWARM.

Fuente: Propia

También se requiere de una herramienta que sea capaz de manipular y procesar este tipo de data, por ello, es este trabajo utilizaremos una librería de *Python* llamada "*Obspy*". La librería posee herramientas de análisis para formatos usados en sismología (entre ellos *miniSEED*), la librería provee de funciones, clientes para el acceso a *Data Centers* y rutinas de procesamiento de señal que permite la manipulación de series de tiempo sismológicas o infrasónicas [35].

3.2.3. Medidas para Mitigar el Ruido en la Señal de Infrasonido

La mitigación del ruido requiere que se considere los distintos tipos de ruido por separado:

El ruido de tipo electrónico puede ser despreciable si tomamos en cuenta que las señales que nos interesan son de magnitud mucho mayor, por lo mismo, se ha optado por no tomar en cuenta para este trabajo señales muy débiles que pueden tratarse de señales muy lejanas o fuera del rango de 3 Km.

El ruido de tipo electromagnético puede generar data errónea en la data entregada por el sensor *Raspberry Boom* (ver figura 30), esto se da a causa de que este sensor es muy

susceptible a la energía de alta frecuencia, este un problema que puede ser solucionado simplemente alejando al sensor de dichas fuentes de energía, el fabricante del sensor *Raspberry Boom* en su página [36] explica que, para que una señal induzca ruido electromagnético en la data del sensor, esta debe tener una frecuencia por encima de 2 GHz y tener una densidad de energía de 50 dB por encima del piso de ruido, por ejemplo, un cable *ethernet*, transmitiendo ondas en las bandas de 2.4 a 2.5 GHZ y 4 a 4.1 GHz aproximadamente, tiene un nivel de densidad de energía de 45 dB sobre el piso de ruido a una distancia de 1.5 m de distancia, por lo que esta debería ser una distancia segura para alejar el sensor. Aún así, existe una fuente de ruido electromagnético que no podemos alejar, debido a que se encuentra embebida, este es el caso de la antena WiFi (*access point*) de la misma *Raspberry Pi*, para este caso la empresa *Raspberry Shake* recomienda no utilizar esta funcionalidad del *Raspberry Pi*, y en caso de querer hacer uso de una conexión WiFi de internet, utilizar una antena de Wifi externa que se encuentre alejada de la placa del sensor *Raspberry Boom*, para este trabajo se decidió utilizar una antena WiFi USB externa alejada del sensor por medio de un extensor Hub de puertos USB que se conecta al *Raspberry Pi*.

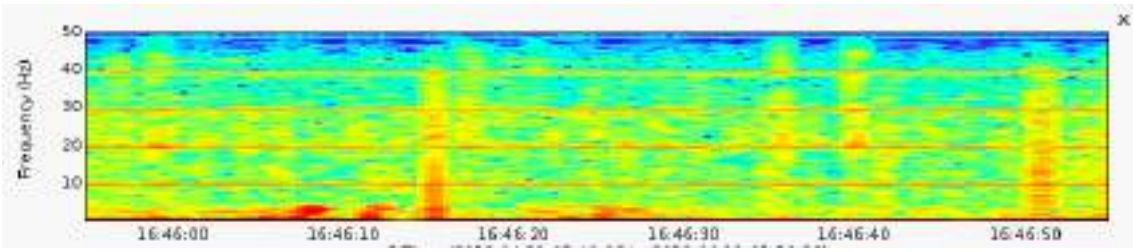

Figura 30: Mapa de calor de los componentes de frecuencia vs. tiempo de una señal (Espectrograma). Este representa a una señal con ruido electromagnético causado por energía en la banda de 2 GHz y bandas superiores.

Fuente: [36]

El ruido de tipo acústico que se da en la banda de infrasonido es el que más afecta las señales obtenidas por el sensor. Por un lado, debido a que la laguna Palcacocha se encuentra en una zona de poco acceso, podemos esperar que no exista ruido de automóviles ni ningún otro ruido relacionado al entorno urbano, pero por otro lado, aunque no es muy frecuente, es posible de vez en cuando escuchar aviones pasando cerca de la zona de interés por lo que este es un ruido a considerar. Como ruidos de tipo acústico también tenemos fenómenos

meteorológicos, siendo uno de ellos, el del trueno, cuyo sonido es muy parecido al de las avalanchas. Todos estos ruidos de tipo acústico se intentarán clasificar para diferenciarlos de las avalanchas por medio del procesamiento de señales con algoritmos de *Machine Learning*. Otro tipo de ruido es el generado por el viento, que como se explicó en la sección 2.2.1.1, hace prácticamente imposible obtener mediciones precisas ya que puede generar ruido de magnitudes muy grandes, cubriendo cualquier señal que deseáramos medir.

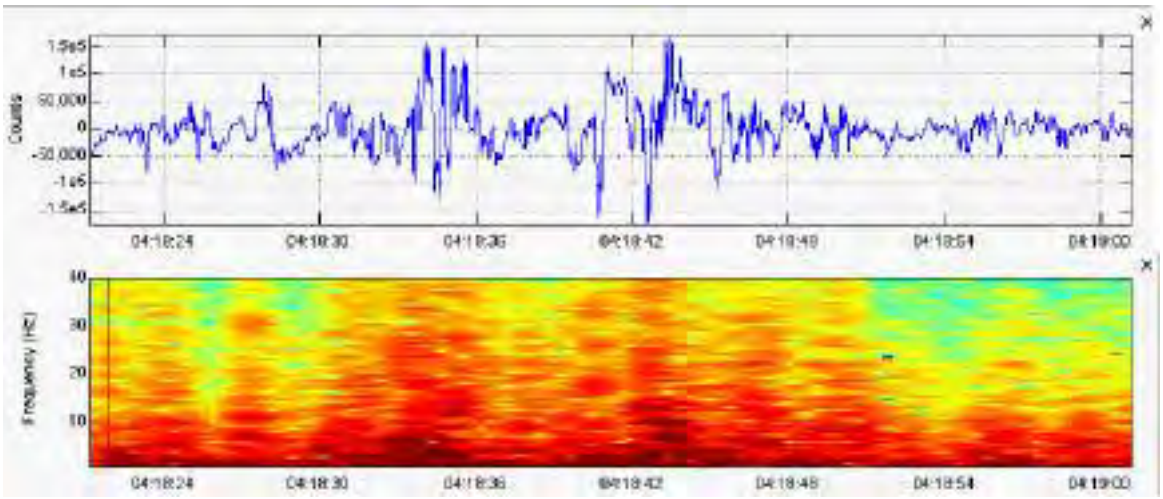

Figura 31: Señal que representa la medición del viento en un sensor de infrasonido.
La parte superior representa la señal en el dominio del tiempo y la parte inferior, su espectrograma.

Fuente: Propia

Cuando el sensor se encuentra protegido naturalmente por extensiones de bosques, el problema del viento disminuye considerablemente, sin embargo, en la laguna Palcacocha no existen bosques cercanos de tales características. Existen métodos muy confiables de reducción de ruido que usan correlación multicanal a las señales obtenidas por medio de un arreglo de varios sensores [27], lamentablemente debido a que para este trabajo solo se dispone de un único sensor, debemos buscar otros métodos.

El método que se ha decidido utilizar es el uso de pantallas de viento o *windscreens*, este método fue estudiado por un estudio de investigación donde se comparó la efectividad del uso de *windscreens* para la medición de infrasonido [18], el principio detrás del uso de *windscreens* es que, como se vio en la sección 2.1.1, el sonido a bajas frecuencias tiene un coeficiente de absorción mucho menor, el infrasonido por tanto, tiene una atenuación mucho

menor no solo en su propagación en el aire sino también en su propagación a través de distintos materiales, en otras palabras, el infrasonido puede traspasar materiales con mayor facilidad. Son cuatro los requerimientos para que un *windscreen* cumpla su papel de forma óptima: Gran reducción del ruido de generado por el viento, alta transmisibilidad del sonido por debajo de los 20 Hz, ausencia de tonos eolianos (ruido generado por el paso del viento a través de los objetos, en inglés se le llaman "*Aeolian Tones*") y una baja retención de agua (requerida para su funcionamiento en todos los climas, incluido climas lluviosos). En la siguiente figura se hizo un análisis experimental de la efectividad para la reducción del ruido causado por el viento en tres materiales.

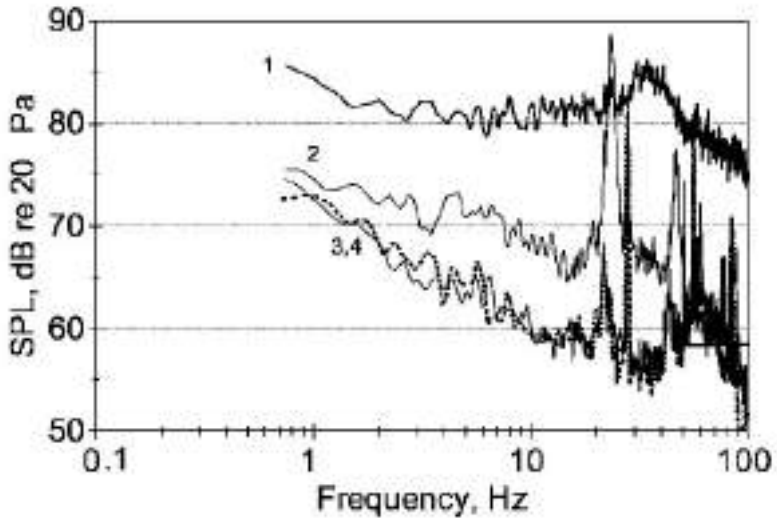

Figura 32: Nivel de presión acústica del ruido del viento medido por un micrófono que se encuentra dentro de *windscreens* de distintos materiales. Todos los *windscreens* tienen un diámetro interno de 7cm y un espesor de sus paredes de 1.27cm. La velocidad del viento fue de 9.3 m/s. Leyenda: (1) Sin ninguna *windscreen*, (2) Material hecho de loseta de transbordador espacial, (3) Madera balsa (gráfica punteada) y (4) Espuma de poliuretano de celda cerrada.

Fuente: [18]

Como se observa, los materiales que mejor rendimiento tienen para la reducción del ruido de viento son la espuma de poliuretano y la madera balsa. En vista de la fácil disponibilidad y precio accesible de la madera balsa, se decidió utilizar este material para el diseño de el *windscreen* que se utilizará en este trabajo.

El sensor *Raspberry Boom* en si mismo ocupa más de 7cm, por lo que el espacio interno será un poco más grande, pero el grosor de las paredes del material si deben tener 1.27cm de espesor. La forma del *windscreen* será una caja.

Figura 33: Ejemplo de una caja para el *windscreen*, en su interior se colocará el sensor de infrasonido. El material será madera balsa, las paredes deben ser de un espesor de 1.27cm y debe tener las siguientes dimensiones internas: una base de 13cm x 16cm y una altura de 8.5cm.

Fuente: https://disbaltienda.com/product/caja-de-balsa/

Como paso final para el diseño de la *windscreen*, la caja deberá ser pintada con una fina capa de barniz a fin de que pueda protegerse del sol, y del agua de la lluvia.

Es importante añadir, que este método solo atenúa los ruidos producidos por el viento, no los elimina, señales de ruidos de viento lo suficientemente grandes pueden traspasar el *windscreen* y producir señales de ruido que pueden esconder fácilmente las señales de avalancha. Para una reducción más efectiva de los ruidos de gran magnitud producidos por viento se requieren métodos que implican el uso de arreglos de sensores, los cuales se encuentran fuera del alcance de este trabajo de investigación ya que, para el sistema propuesto, solo se dispone de un solo sensor.

3.2.4. Fragmentos de señal para procesamiento en tiempo real

Para el análisis en tiempo real, elegiremos una porción de la señal o ventana de longitud especifica para ser analizada cada cierto tiempo especifico. Por ejemplo, como el sistema recibe información en tiempo real, esperamos llenar una ventana con una cierta cantidad de muestras, hasta lograr una longitud fija y cerramos la ventana, mientras aplicamos el algoritmo de detección de avalanchas en esta ventana, vamos llenando la siguiente ventana de la misma longitud, de esta forma seguimos este procedimiento de forma sucesiva como un bucle infinito. Aunque técnicamente sería correcto llamar como "ventanas" a estas porciones de señal; para no confundir con la técnica de "enventanado" o "*windowing*", los llamaremos "fragmentos" de ahora en adelante.

Lo primero a tomar en cuenta para diseñar los fragmentos de análisis, es decidir si estos se seguirán entre sí de forma continua o si existirá una superposición entre ellos. Cuando los fragmentos son continuos, solo pueden comenzar a llenarse cuando el anterior fragmento se ha terminado de llenar. Si los fragmentos tienen superposición, entonces un fragmento puede comenzar a llenarse cuando el anterior aún sigue en proceso de llenado. Así como los fragmentos deben ser de longitudes fijas, la superposición también debe ser de un tamaño fijo para todos los casos. Los fragmentos contiguos se ilustran mejor en la siguiente figura:

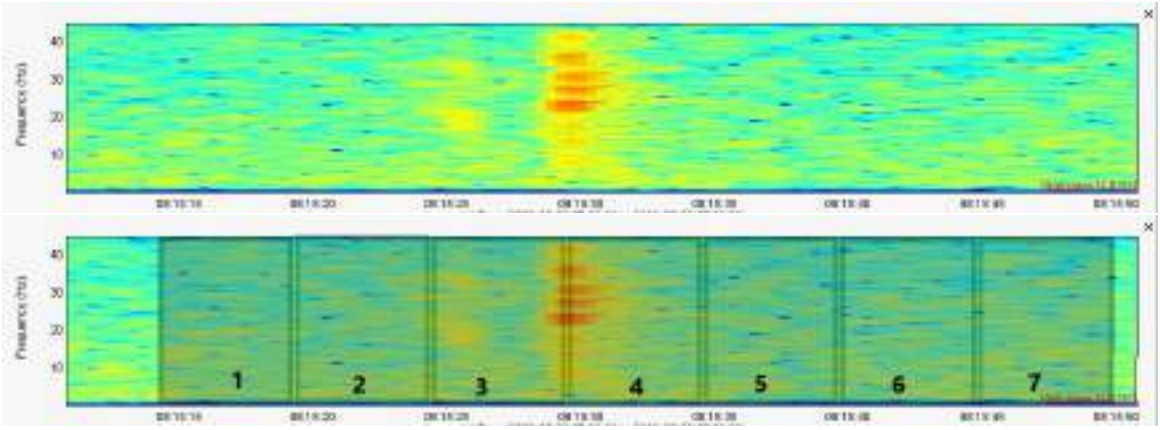

Figura 34: La primera figura corresponde al espectrograma de una avalancha cuya señal de infrasonido tiene una duración aproximada de 3.5 segundos. La segunda figura corresponde a la misma señal pero separada en fragmentos contiguos de 5 segundos.

Fuente: Propia

Para elegir si usaremos fragmentos superpuestos o contiguos, debemos tener en cuenta

que estos últimos tienen un problema muy notorio, y es que, como se puede observar en la muestra de avalancha de la figura 34; ya que la data llega en tiempo real, los fragmentos se acomodan de forma fija en intervalos constantes y existe una posibilidad de que, incluso si el fragmento es más grande que un evento de avalancha, este no sea capaz de cubrir por completo este evento en un solo fragmento (notar que la señal de avalancha se encuentra cubierta por los fragmentos 3 y 4). Por otro lado, los fragmentos superpuestos tienen la ventaja de que un mismo evento puede ser analizado en varios fragmentos, incrementando así la posibilidad de que un fragmento cubra una avalancha de forma completa, la única desventaja frente a los fragmentos contiguos es que se requiere más esfuerzo computacional ya que las ventanas serán analizadas de forma mucho más frecuente mientras más sea la longitud de la superposición. Los fragmentos superpuestos se ilustran mejor en la siguiente figura:

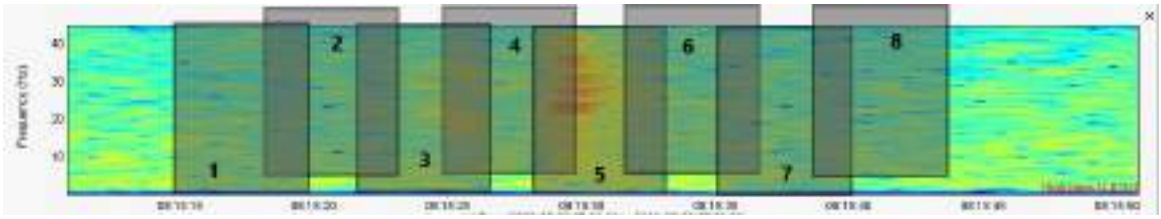

Figura 35: Separación de un evento de avalancha en fragmentos superpuestos, se posicionó a los fragmentos pares más alto para notar mejor la diferencia entre todos. Los fragmentos son de 5 segundos y la superposición es de 1.67 segundos. Notar que el fragmento 5 cubre por completo la avalancha.

Fuente: Propia

En vista de lo explicado en el anterior párrafo se ha decidido utilizar fragmentos superpuestos, por lo que, lo siguiente sería elegir la longitud de estos fragmentos y ya que son del tipo superpuesto, también debemos determinar la longitud de la superposición.

A fin de elegir la longitud de fragmento y superposición más óptimos, es pertinente señalar las desventajas de escoger fragmentos muy pequeños y muy grandes. Los fragmentos pequeños tienen la desventaja de que existen muchas otras señales que no son avalanchas que también son cortas como se puede ver en la figura 36. Además de estos ruidos coherentes, el viento es también un ruido a tomar en cuenta, como se explicó en la sección 2.2.1.1, los ruidos producidos por el viento no pueden ser mitigados completamente. Las señales de

ruidos de viento pueden ser muy impredecibles, de duración variada y, de hecho, son en su gran mayoría de longitud corta. En cantidad de eventos, a lo largo del día, es obvio que la cantidad de eventos de no-avalancha superan ampliamente a los eventos de avalanchas, por lo que incluso si el algoritmo de detección tiene un 99 % de efectividad, ese 1 % de errores puede llevar a una gran cantidad de falsos positivos si es que hacemos una clasificación cada segundo durante 24 horas.

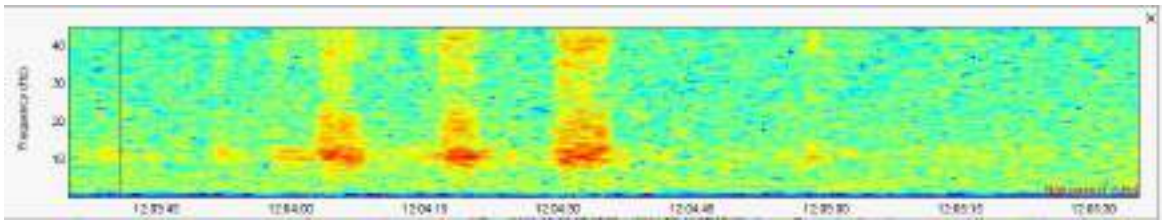

Figura 36: Espectrograma de varios eventos pequeños de ruido que corresponden a un vuelo de avión que pasó cerca a la ubicación del sensor de infrasonido

Fuente: Propia

En contraste, si elegimos fragmentos grandes, podemos usar a favor las características en tiempo del fragmento para entrenar nuestro algoritmo, de forma que no considere como avalanchas a los eventos que sean de duración menor a la longitud del fragmento. De esta manera, aunque perdemos las señales de avalancha débiles o cortas, filtramos muchísimos más eventos de no-avalanchas que podrían potencialmente generar errores de falsos positivos en la clasificación. Mientras más grande sea nuestro fragmento más fácil será discriminar los eventos cortos, que aunque más numerosos, son mucho menos importantes que los eventos duraderos y grandes, que finalmente son las avalanchas que pueden generar algún potencial desastre. No obstante las ventajas mencionadas antes, usar fragmentos demasiado grandes no es buena idea, y es que, durante el tiempo que tarda en llenarse un fragmento, no se hace ningún procesamiento ni detección, por lo que si este tiene un tamaño de 20 segundos (2000 muestras), entonces se generaría un retraso de al menos 20 segundos (sin contar otro tipo de retrasos) desde que el sensor comienza a recibir las señales correspondientes a un evento (avalancha o ruido) hasta que el algoritmo pueda procesar dicho fragmento y determinar de que evento se trata.

Según todo lo explicado anteriormente, se decidió usar fragmentos de análisis de 10.24

segundos (1024 muestras), debido a que, aunque la duración promedio de las avalanchas más significativas es de 20 segundos [14], se decidió que se quería cubrir algunas otras avalanchas de menor duración, un poco menos significativas, pero que se pudieron observar cerca a la laguna Palcacocha; y además de que, un retraso de 20 segundos, solo en esta etapa, es demasiado grande para un sistema de detección que se espera sea de respuesta rápida. En lo que respecta a la longitud de la superposición, se decidió que sea de 8.24 segundos, lo que significa que, se tendrán fragmentos de análisis listos cada 2 segundos, y por tanto, el algoritmo podrá realizar la detección en tiempo real a una velocidad de una clasificación cada 2 segundos. Tener en cuenta, que esta longitud no solo es importante para el análisis en tiempo real, sino también para el tamaño de las muestras que serán recolectadas para el dataset de entrenamiento del modelo; es decir, el entrenamiento debe ser hecho con muestras de 10.24 segundos.

3.2.5. Segmentación y Enventanado

Para este trabajo es más importante el análisis en frecuencia que el análisis temporal, es decir, que aunque es importante el análisis del desarrollo en el tiempo de una avalancha, no es necesario que la segmentación sea tan detallada como para requerir una muy alta resolución temporal. En este sentido, se decidió utilizar segmentos con una longitud de 256 muestras (2.56 segundos).

En lo que respecta al enventanado, se decidió utilizar una longitud de solapamiento del 50 % y una función Hann para la ventana, siendo ambos criterios muy comunes en el procesamiento de audio.

De esta forma, los criterios que se utilizaran en el presente trabajo, en lo que respecta a segmentación y eventanado, quedan de la siguiente forma:

- Longitud de los *frames*/segmentos: 2.56 segundos (256 muestras)
- Longitud del solapamiento: 1.28 segundos (128 muestras)
- Tipo de ventana utilizada: Ventana Hann

Con estas longitudes de segmentos y solapamiento, y conociendo que la longitud de un fragmento de análisis es de 10.24 segundos (ver sección 3.2.4), entonces la cantidad de

ventanas que corresponderían a un fragmento serían:

$$NumeroVentanasPorFragmento = \frac{Long.Fragmento - Long.Solapamiento}{Long.Frames - Long.Solapamiento}$$

$$NumeroVentanasPorFragmento = \frac{10.24 - 1.28}{2.56 - 1.28} = 7\ ventanas$$

3.3. Diseño del sistema

El diseño del sistema propuesto incorpora siete principales módulos: el módulo de recolección de datos, el módulo de pre-procesamiento, el módulo de extracción de características, el módulo de entrenamiento del modelo, el módulo de análisis en tiempo real, el módulo de clasificación de eventos y el módulo de visualización del resultado. El sistema se compone de dos etapas que pueden compartir algunos módulos, en la figura 37 se ilustra la arquitectura de alto nivel del sistema completo.

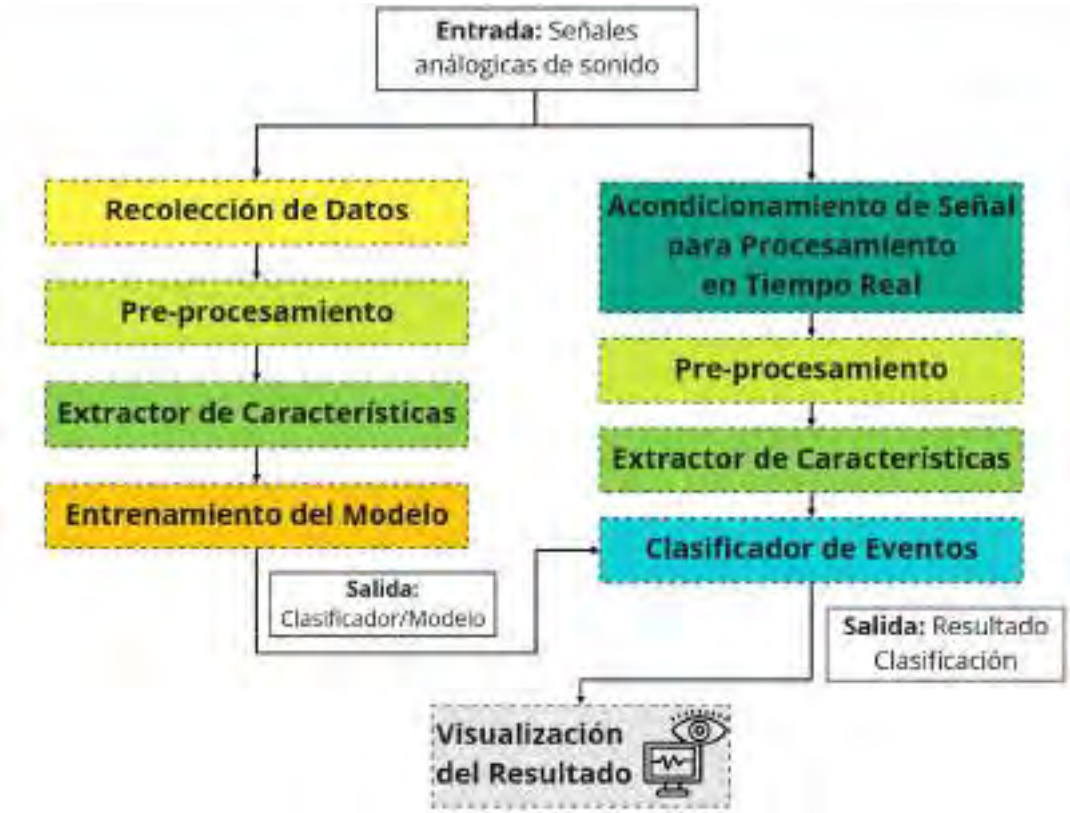

Figura 37: Arquitectura de alto nivel del sistema propuesto

Fuente: Propia

En las siguientes secciones se explicará en detalle el diseño y funcionamiento de los seis módulos que componen el sistema.

3.3.1. Módulo de Recolección de Datos

Este módulo esta compuesto de 3 principales pasos: Sensado y digitalización de la señal, transmisión por la red y generación del *Training Dataset*. La figura 38 representa la arquitectura del módulo propuesto.

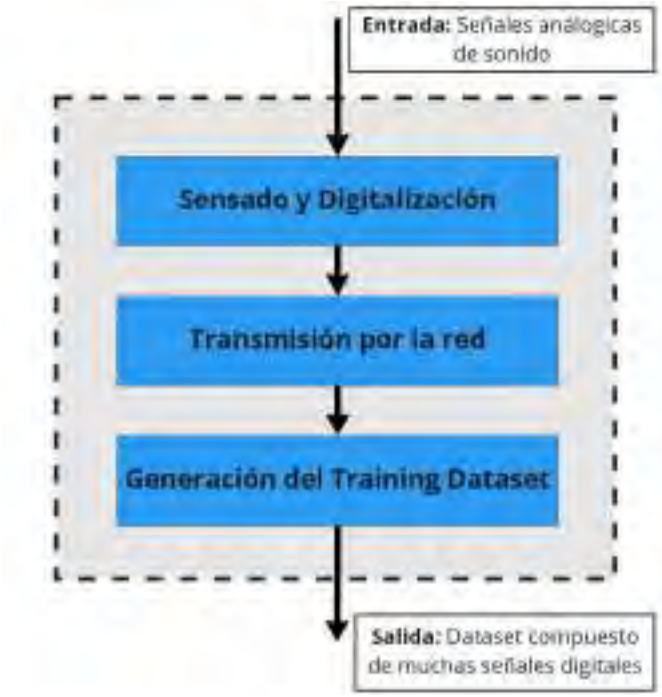

Figura 38: Arquitectura del módulo de recolección de datos.

Fuente: Propia

3.3.1.1. Sensado y Digitalización de la Señal

Los procesos de Sensado y Digitalización suceden en el circuito embebido del sensor microbarógrafo *Raspberry Boom* que se acopla al *Raspberry Pi*.

El sensor microbarógrafo transforma la señal analógica de sonido en una señal analógica de voltaje, este sensor electrónico puede estar afectado por distintos ruidos relacionados al sonido, diferencias de presión y fenómenos meteorológicos, es necesario instalar el sensor en una ubicación lo mejor protegida de la lluvia y el viento. Debemos tener en cuenta también que las avalanchas generan sonidos que se van atenuando mientras más lejos se propagan,

por lo que, el sensor debe encontrarse lo más cerca posible (menor a 3 Km [9]) a los nevados donde suceden las avalanchas. El único método físico que se utilizará para la reducción del ruido de viento será la caja de madera balsa, cuyo funcionamiento se explicó en detalle en la sección 3.2.3.

Además de la ubicación del sensor, también debemos tener en cuenta su alimentación de energía. Gracias a que el sensor se encuentra acoplado al *Raspberry Pi*, el sensor será energizado con la misma alimentación que se usa para el *Raspberry Pi*, la alimentación óptima del ordenador *Raspberry Pi* debe ser de 5V a 2.5A en corriente continua (ver anexo 01), por lo que cualquiera que sea la fuente de alimentación debe cumplir como mínimo con estos valores.

Para la digitalización, el primer paso es pasar las señales analógicas de voltaje obtenidas por el microbarógrafo a un formato digital; todo este procedimiento lo realiza un circuito que ya se encuentra embebido en el sensor de infrasonido *Raspberry Boom*. En el proceso de digitalización, la señal se cuantifica en el tiempo a una determinada frecuencia de muestreo y la amplitud se cuantifica a una cierta profundidad de bits; en el anexo 02 podemos observar que las especificaciones del digitalizador son las siguientes: La tasa de muestreo es de 100 muestras por segundo (100 Hz) y la profundidad de 24 bits. La resolución temporal esta determinada por la tasa de muestreo pero la resolución en amplitud del audio se mide en bits, 24 bits quiere decir que la señal de audio puede registrar hasta 2^{24} (16,777,216) valores discretos para los niveles de intensidad sonora (o un rango dinámico de 144 dB).

Al final del proceso de digitalización, el sensor almacena la data de infrasonido en la memoria SD del *Raspberry Pi* en el formato *miniSEED*.

3.3.1.2. Transmisión por la Red

Para poder trabajar con las señales digitales de forma remota, debemos poder conectar el *Raspberry Pi*, al que esta acoplado el sensor, a una red de internet. Afortunadamente, en la laguna Palcacocha existe una antena que provee una red WiFi de internet (ver sección 3.1) y que se conecta a la sede del INAIGEM en Huaraz por medio de un radioenlace. Lo siguiente sería poder conectar el *Raspberry Pi* a esta red, para ello la primera opción sería utilizar el módulo WiFi del mismo *Raspberry Pi*, pero como se explicó en la sección 3.2.3, para evitar la interferencia de ruidos electromagnéticos lo mejor es utilizar un módulo USB externo de

WiFi con un extensor. Para poder ubicar este módulo externo WiFi, solo debemos tener en cuenta que tenga línea de vista con la antena que provee red WiFi en la laguna y que se encuentre protegido de la lluvia y el sol.

Una vez que el *Raspberry Pi* se haya conectado a internet podemos acceder a su data de forma remota, siempre y cuando nos encontremos en la misma red, esto podemos hacerlo a través de una conexión SFTP al ip asignado o a través de la interfaz de usuario ingresando a la URL "rs.local" (según se puede ver en la documentación del sensor [36]). Después de conectarnos, podemos descargar la data, o bien copiando los archivos mediante el protocolo SFTP o bien a través de la interfaz de usuario. Según la documentación del sensor, este puede almacenar hasta 7 días de data (configurable) y cada día se genera un archivo *miniSEED* de aproximadamente 15 mb.

3.3.1.3. Generación del Training Dataset

Antes de comenzar a generar el Dataset de entrenamiento, requerimos de dos elementos básicos: la data de infrasonido y un registro objetivo de los eventos; el primer elemento es el que obtuvimos en la anterior sección, el segundo es un registro del tiempo exacto en el que inicia cierto evento de infrasonido y el tiempo que finaliza, este registro lo obtendremos de una observación visual de las avalanchas en la laguna Palcacocha, para ello nos apoyaremos de las grabaciones en vídeo que se obtuvieron del sistema de monitoreo que existe en la laguna (ver sección 3.1). Este registro se hará de forma manual, observando las grabaciones y registrando correctamente los tiempos y la duración de los eventos de avalancha, apoyándose también de las señales que se observen en la data de infrasonido. Al momento de registrar la hora de un evento, se debe tener en cuenta que, la observación visual de una avalancha es inmediata ya que la luz viaja a la velocidad de la luz (300,000 Km/s), pero la percepción sonora es muchísimo más lenta, ya que la velocidad del sonido en el aire es muchísimo menor (340 m/s); es decir, aunque se haya observado una avalancha a una hora determinada, la señal de sonido que genere llegará al sensor con un retraso que se incrementará mientras más lejos se encuentre el sensor del lugar donde sucedió la avalancha.

Luego, para la generación del *Training Dataset* se debe seguir los siguientes pasos: Utilizar los registros de los eventos reales para extraer las señales de infrasonido, limitarlas a muestras de 10.24 segundos (ver sección 3.2.4), exportarlas a archivos csv y finalmente almacenarlas separándolas por clase. Para este proyecto, la data se debe separar en dos

clases: la primera clase corresponderá a las señales correspondientes a todas las avalanchas registradas y se llamará “Avalanchas” y la segunda clase corresponderá a todos los eventos que no son avalanchas, se utilizará una sola clase llamada “No-avalancha”, en esta última clase se incluyen todos los eventos en el rango del infrasonido que son considerados ruidos (viento, aviones, motores, truenos, etc.).

Una de las tareas fundamentales del presente proyecto es recoger la mayor cantidad de muestras de avalanchas que sea posible ya que es imposible encontrar en internet un Dataset de infrasonido de eventos de avalanchas. Al momento de armar el Dataset, se debe tener en cuenta que el número de muestras de avalanchas y no-avalanchas debe ser igual o similar; por un lado, las avalanchas son eventos de ocurrencia escasa, y por otro lado, los eventos de no-avalancha ocupan la gran mayoría de eventos que suceden durante el día, por lo que debemos aprovechar lo más que se pueda de las muestras de avalancha y también escoger cuidadosamente una selecta variedad de ruidos para evitar el *overfitting*, de esta forma podremos tener una cantidad equilibrada de muestras en el *Training Dataset*.

Resumiendo, el *Training Dataset* debe tener las siguientes características:

- 2 Clases: “Avalanchas” y “No-avalanchas”
- Cantidad de muestras de Avalanchas debe ser igual o similar a las de No-avalanchas
- Cantidad recomendada de muestras de avalanchas: 50 o más
- Cantidad recomendada de muestras de no-avalanchas: 50 o más
- Duración de todas las muestras: 10.24 segundos
- Formato de almacenamiento: csv

3.3.2. Módulo de Pre-Procesamiento

Este módulo esta compuesto de 3 principales pasos: un filtro pasabanda, un filtro de rechazo de banda y un filtro de reducción de ruido. La figura 39 representa la arquitectura del módulo propuesto.

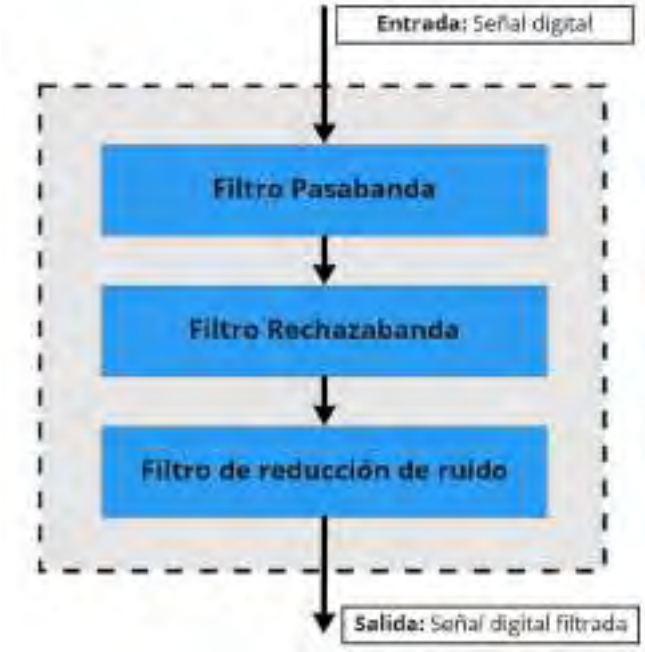

Figura 39: Arquitectura del módulo de pre-procesamiento.

Fuente: Propia

3.3.2.1. Filtro Pasabanda

Las avalanchas tienen componentes desde 0 Hz hasta los 45 Hz, tal como se ve en la figura 40, por lo que en teoría no se requeriría ningún filtro, sin embargo, debido a que los ruidos generados por viento tienen gran presencia en la banda más baja del espectro (0Hz - 2Hz), se decidió filtrar todas las señales por debajo de frecuencias cercanas a 2 Hz, y para no perder del todo la información de baja frecuencia de las avalanchas se decidió utilizar un límite inferior de 1.8 Hz en el filtro. En lo que corresponde al límite superior, se decidió utilizar 45 Hz, ya que se notó en una revisión de algunas muestras de avalancha y de trabajos anteriores que las avalanchas pueden tener componentes hasta esa frecuencia [14][12].

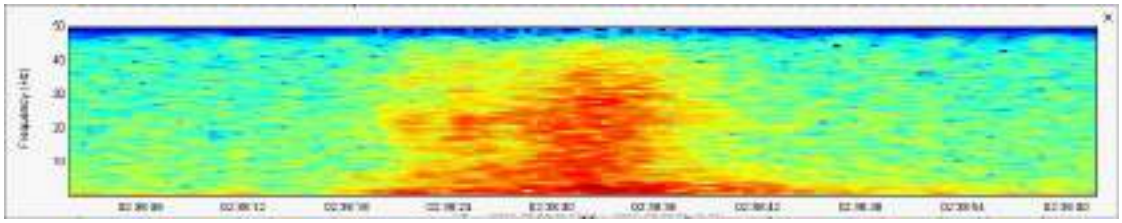

Figura 40: Espectrograma de una muestra de avalancha en el infrasonido.

Fuente: Propia

De esta forma el filtro pasabanda se diseñó con la siguiente configuración:

- Tipo de filtro: Filtro Butterworth Pasabanda
- Frecuencia de corte inferior: 1.8 Hz
- Frecuencia de corte superior: 45 Hz
- Orden del filtro: 6
- Tasa de Muestreo: 100 Hz

La orden del filtro se eligió basado en un análisis en la frecuencia de estos filtros bajo diferentes filtros con distintas ordenes, además de evitar que sea de un orden muy alto, ya que requiere mas esfuerzo computacional.

3.3.2.2. Filtro Rechaza Banda

En la ubicación donde se instalará el sistema existe un ruido coherente y relativamente constante, que es el de un motor de motobomba, este motor es encendido periódicamente por el personal del gobierno regional para extraer agua de la laguna durante un par de horas del día, se ha observado que este motor genera un ruido en la banda de 25 a 30 Hz (ver figura 41), por lo que se decidió usar un filtro rechaza banda que mitigue las señales en esta banda.

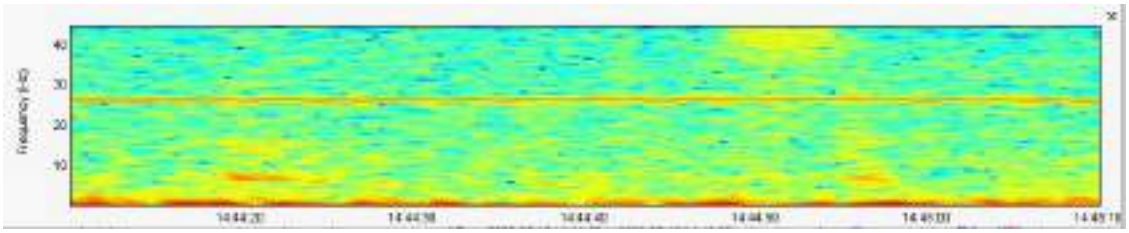

Figura 41: La línea que se observa en los 26.5 Hz es el ruido que genera el motor motobomba encendido

Fuente: Propia

De esta forma el filtro rechaza banda se diseñó con la siguiente configuración:

- Tipo de filtro: Filtro Butterworth Rechaza Banda
- Frecuencia de corte inferior: 25 Hz

- Frecuencia de corte superior: 30 Hz
- Orden del filtro: 6
- Tasa de Muestreo: 100 Hz

La figura 42 representa la transformada de Fourier de la señal de avalancha mostrada en la figura 40, como se observa, esta se encuentra sin ninguna clase de filtro. Más abajo, en la figura 43 , se muestra la misma señal pero después de pasar por los filtros de pasabanda y rechazabanda diseñados, se observa que mantiene aún la información más importante mientras filtramos los ruidos de frecuencia baja.

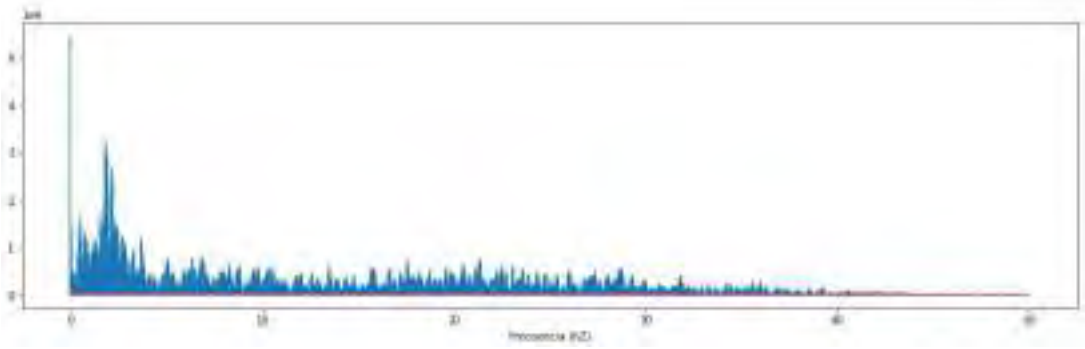

Figura 42: Transformada de Fourier de señal de avalancha antes de aplicar filtros.

Fuente: Propia

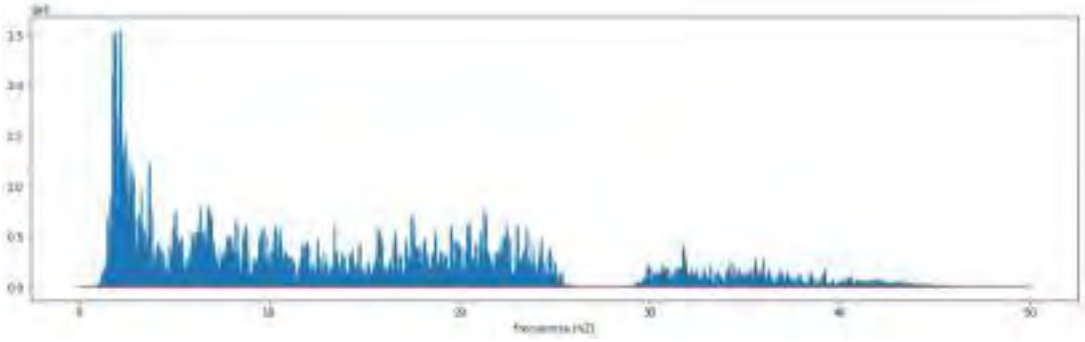

Figura 43: Transformada de Fourier de señal de avalancha después de aplicar los filtros Butterworth pasabanda y rechazabanda.

Fuente: Propia

3.3.2.3. Filtro de Reducción de Ruido

En la etapa final del módulo de pre-procesamiento se intentará reducir el ruido de la señal, para ello se utilizará un filtro de Puerta de Ruido Espectral, el cuál utiliza una técnica novedosa de limpieza de ruido llamada"*Spectral Gating*" que fue explicada en la sección 2.3.3.3, el algoritmo explicado en esa sección fue implementado en una librería por el mismo equipo de investigación y es de código abierto por lo que nos apoyaremos en dicho trabajo para el desarrollo de esta etapa.

Para mostrar el funcionamiento del filtro se hizo una prueba en una señal de avalancha débil, de la cual se observa su espectrograma en la figura 44, cabe añadir, que esta señal se generó en un contexto de casi inexistente viento y sin ningún otro ruido que interfiera.

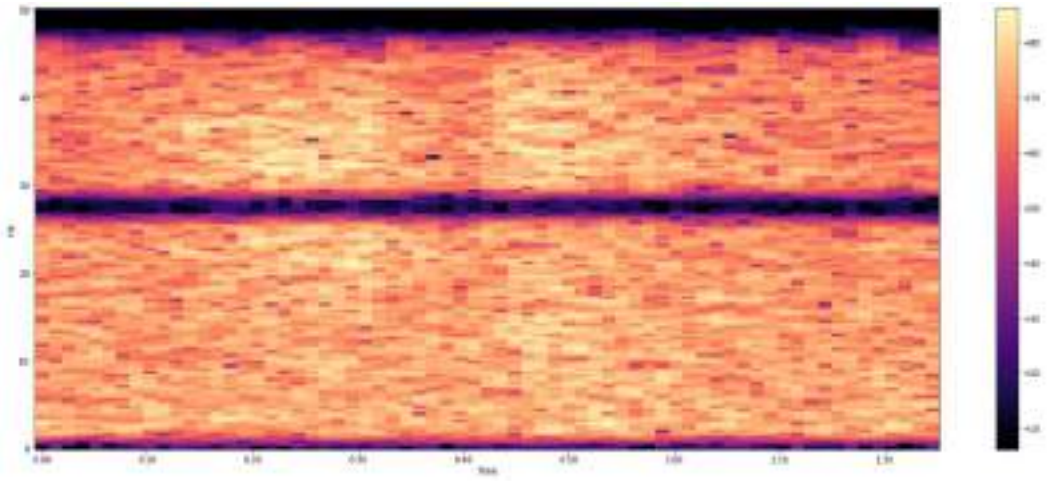

Figura 44: Espectrograma de una señal de avalancha débil al que le fueron aplicados solo los filtros Butterworth pasabanda y rechazabanda.

Fuente: Propia

En la figura 45 se observa el espectrograma de la señal débil después de pasar por todos los filtros del módulo de pre-procesamiento, como se observa, aunque la señal es débil, se pudo recuperar sus características en frecuencia.

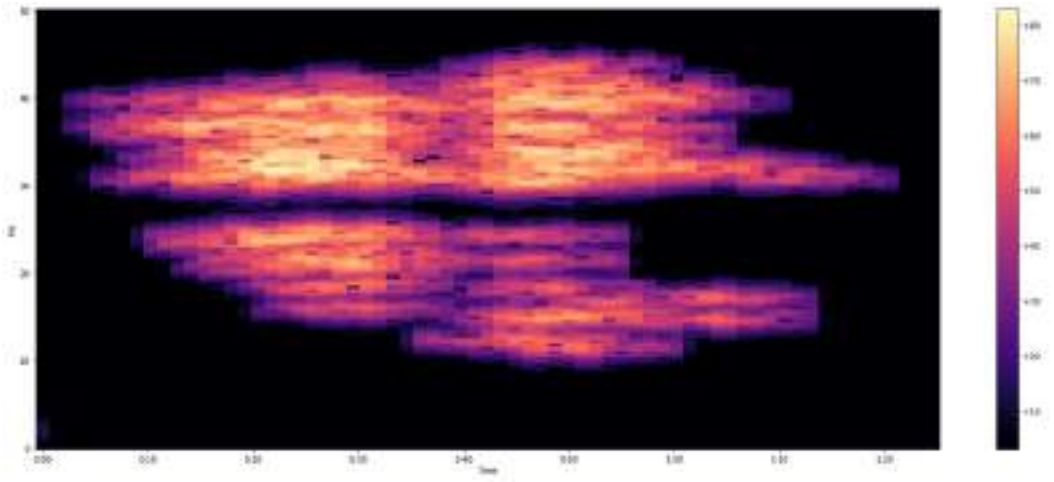

Figura 45: Espectrograma de una señal de avalancha débil al que le fueron aplicados los filtros Butterworth pasabanda y rechazabanda, y el filtro de reducción de ruido.

Fuente: Propia

El algoritmo utilizado requiere distintos argumentos que se le deben pasar para que este pueda funcionar correctamente, el detalle de estos argumentos y sus valores por defecto se pueden observar en la documentación del algoritmo [31]. Los argumentos deben ser ajustados para que el filtro funcione correctamente en las señales de infrasonido. Como el filtro utiliza su propio *framing* y *windowing*, se utilizó 128 muestras para el tamaño de ventana y 56 para el solapamiento, que corresponde a 1.28 segundos, un tiempo no tan pequeño como el recomendado para señales de audio pero como se explicó en la sección 3.2.5, para las señales analizadas es más importante la información en el campo de la frecuencia y no tanto en el tiempo. En lo que respecta a la suavización de la máscara en frecuencia y tiempo, se le debe pasar como argumentos un rango de frecuencia (en Hz) y uno de tiempo (en milisegundos), se eligió 1.8 Hz y 5000 ms respectivamente. Luego se le debe pasar el número de desviaciones estándar sobre el promedio para ubicar el umbral espectral que separará señal y ruido, mientras este valor es más alto, el umbral será más alto por lo que se atenuaran más señales, el valor por defecto es 1.5 pero se eligió 1 para este trabajo ya que se observó que algunas señales débiles eran atenuadas de forma muy agresiva. Además como se explicó más arriba, una muestra de ruido también debe ser pasada como argumento, en nuestro caso tenemos muchos tipos de ruidos los cuales podríamos utilizar, sin embargo se eligió una muestra de “silencio”, es decir, el ruido que genera el mismo sensor cuando no detecta ningún evento, se eligió este tipo de ruido debido

a que es el único que afecta, de forma constante y con la misma intensidad, a todas las señales obtenidas por el sensor, la muestra elegida es una de 20 segundos donde no se detectó ningún evento y el viento era casi inexistente.

De esta forma el filtro de reducción de ruido se diseñó con la siguiente configuración:

- Tipo de filtro: Filtro de Reducción de Ruido Estacionario
- Longitud de ventana: 128 muestras
- Longitud del solapamiento: 64 muestras
- Tasa de Muestreo: 100 Hz
- Número de Desv. Estándar: 1
- Rango de Frecuencia de suavizado de máscara: 1.8 Hz
- Rango de Tiempo de suavizado de máscara: 5000 ms
- Muestra de Ruido: Muestra de 20 segundos de “silencio”

3.3.3. Módulo de Extracción de Características

El siguiente módulo es el módulo de extracción de características, el cual se utilizará en la etapa de entrenamiento del modelo y en la etapa de detección automática en tiempo real. La figura 46 representa la arquitectura del módulo propuesto.

Figura 46: Arquitectura del módulo de extracción de características.

Fuente: Propia

El proceso de extracción de características o *features* es una parte muy importante de todo el sistema de detección, este proceso, en si mismo, no es muy complejo, lo complicado es elegir, que características serán las que se extraerán. Como se explicó en la sección 2.5.3, debemos encontrar un patrón de características que nos permitan diferenciar los eventos de avalanchas de otros eventos. Por ende, el primer paso que debemos seguir para el diseño de este módulo, es la búsqueda del patrón de características.

Existen diferentes tipos de características, y cada una de ellas aporta una información diferente. Si bien, es normal que entre distintas características exista cierta correlación, esto no es deseable, puesto que, si dos características tienen una alta correlación entre sí, estarían aportando la misma información, ocasionando una redundancia. Por el contrario, cuanta menos correlación exista entre las características, más información distinta se tendrá para describir la señal acústica. Es importante evitar la redundancia entre características para que el modelo que entrenemos con este patrón resulte lo más óptimo y eficiente posible.

Cuando se trata de encontrar el patrón de características, es muy común seguir el procedimiento de extraer todas las características posibles de una señal y luego probar cual combinación de ellas producen el mejor resultado posible; este procedimiento, sin embargo, puede consumir mucho tiempo, lo ideal es comenzar la selección probando un patrón de características conocido que ya haya funcionado en trabajos o investigaciones anteriores. Un procedimiento común para evaluar la redundancia entre dos características es representarlas en una gráfico de dispersión o "*scattering*", que consiste en ubicar los valores de cada característica en un gráfico 2D, para después etiquetar los puntos de cada grupo de características para diferencia a que tipo de evento o clase pertenecen, por ejemplo, en la figura 47 se muestra una gráfica de *scattering*, donde se evalúa los valores de las características Centroide Espectral (*spectral centroid*) vs. Entropía Espectral (*spectral entropy*).

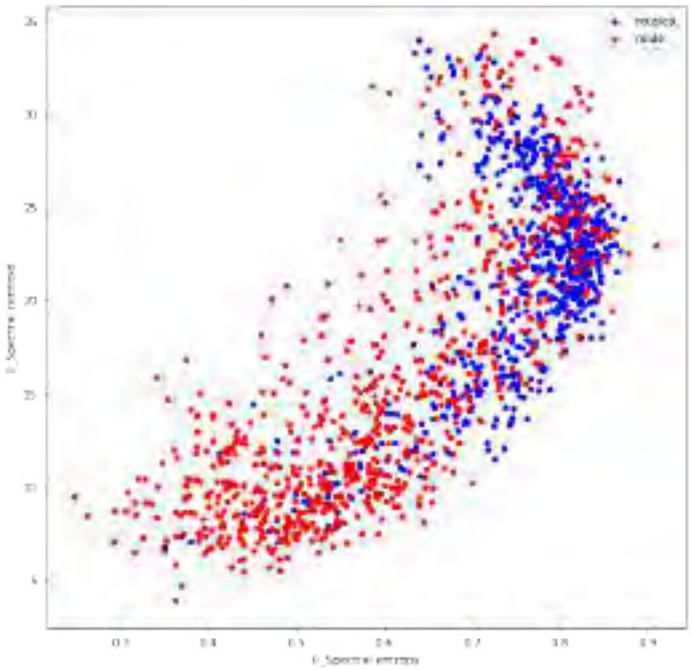

Figura 47: *Scattering*: Centroide Espectral vs. Entropia Espectral.

Fuente: Propia

Como se observa en la figura 47, se trata de una clasificación de música y ruido, ambas clases están representadas por un color, donde los puntos rojos son las características de muestras de ruido y los azules, de música. Aquí se puede apreciar que, estas dos características presentan un nivel pequeño correlación, ya que existen áreas diferenciadas donde se acumulan las muestras de ruido y música, pero también se observan una pequeña cantidad de puntos que se solapan en ambas clases. A simple vista no es posible discriminar las clases al 100 %, sin embargo, debemos recordar que esta es solo una representación bidimensional de dos características, si añadimos una característica más, es posible que se puedan separar las clases en un plano tridimensional.

Por otro lado, podría resultar también que dos características si tengan correlación, en la figura 48 se muestra un *scattering* de las características de Centroide Espectral vs. Roll-off Espectral, de las mismas muestras de música y ruido.

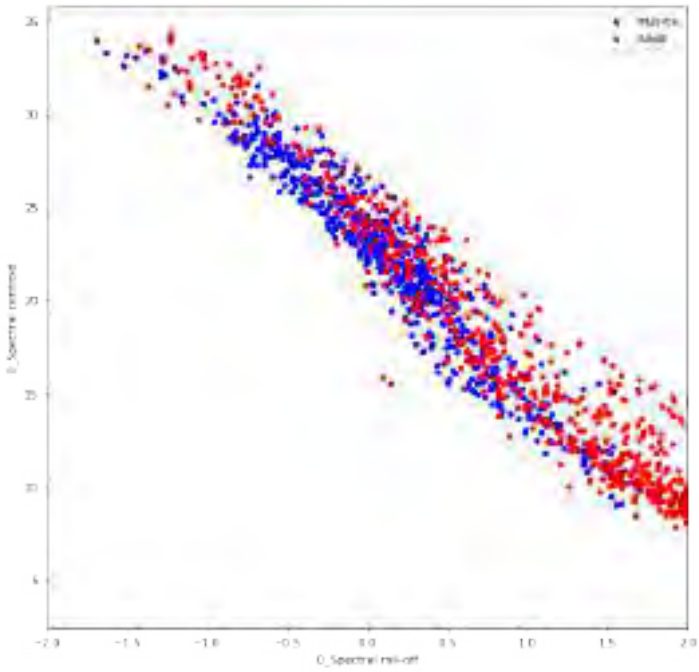

Figura 48: *Scattering*: Centroide Espectral vs. Roll-off Espectral.

Fuente: Propia

En el ejemplo de la figura 48, se observa que existe demasiado solapamiento entre estas dos características, lo cual nos indica que, si bien alguna de ellas pudiera aportar información útil para discriminar las dos clases en otras circunstancias, no es recomendable utilizar ambas porque aportan la misma información, es decir, estas dos características tienen demasiada correlación.

Según lo explicado hasta ahora, para poder comenzar la selección del patrón de características, se evaluaran primero las características que ya funcionaron en alguna investigación anterior [14], estas son las siguientes: Centroide Espectral, Kurtosis Espectral, Envergadura Espectral, Relación entre Energía de Bandas en baja frecuencia (Energía banda 6-8 Hz relativa a la energía total), Relación de Energía entre Bandas en alta frecuencia (Energía banda 26-35 Hz relativa a la energía total) y Relación de Energía entre Bandas de Frecuencia (Energía banda 7 a 16 Hz relativa a la energía banda 16-37 Hz). Además de estas características, se añadirán a la lista de evaluación, algunas características que podrían aportar información relevante: Tasa de Cruce por Cero, Puntos de Giro Positivos Espectrales, Roll-off Espectral, Asimetría Espectral y Pendiente Espectral. Es importante notar que las tres características de “Relación de Energía entre Bandas”, en

realidad pertenecen a un solo tipo de característica que se halla de la misma forma, pero de la que se puede obtener distintos valores según las bandas de frecuencia que se elijan para encontrar su relación de energía.

El resultado de este módulo debería ser un vector que este compuesto de los valores de cada característica extraída, sin embargo, en este proyecto, los fragmentos de análisis tienen una longitud de de 10.24 segundos y cada fragmento está compuestos de 7 ventanas (ver sección 3.2.5), por lo que a cada ventana le corresponde su propio vector de características, en consecuencia, lo que se obtiene de una muestra/fragmento de señal no es un vector sino una matriz de características de tamaño $7 \times n$, donde n es el número de características elegidas (ver figura 49).

Matriz de características de una muestra					
	Característica 1	**Característica 2**	**Característica 3**	**...**	**Característica n**
Ventana 1	V1C1	V1C2	V1C3	...	V1Cn
Ventana 2	V2C1	V2C2	V2C3	...	V2Cn
Ventana 3	V3C1	V3C2	V3C3	...	V3Cn
Ventana 4	V4C1	V4C2	V4C3	...	V4Cn
Ventana 5	V5C1	V5C2	V5C3	...	V5Cn
Ventana 6	V6C1	V6C2	V6C3	...	V6Cn
Ventana 7	V7C1	V7C2	V7C3	...	V7Cn

Figura 49: Matriz de características de una muestra.

Fuente: Propia

Aunque la matriz de la figura 49 es representativa de la señal de entrada, lo que se requiere como salida de este módulo es un vector y no una matriz, esto debido a que el entrenamiento se hace basado en vectores de características, por ello, lo que corresponde es reducir la dimensión de la matriz para convertirla en un vector, esto se hace por medio de un proceso llamado "aplanamiento" o "*flattening*", lo que resulta es un vector de características, que es finalmente, la salida de este módulo. El vector de características resultante es de longitud $7n$ y representa la información más relevante que se pudo extraer de un fragmento/muestra. El procedimiento de *flattening* se observa en la imagen 50.

Figura 50: Procedimiento de "aplanamiento" o"*flattening*" de una matriz.

Fuente: Propia

3.3.4. Módulo de Entrenamiento del Modelo

Este módulo esta compuesto de 2 principales pasos: Entrenamiento del modelo y Evaluación del modelo. La figura 51 representa la arquitectura del módulo propuesto.

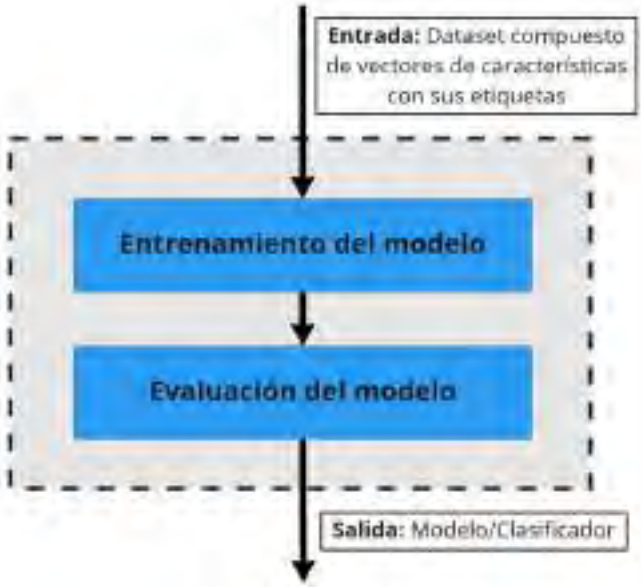

Figura 51: Arquitectura del módulo de entrenamiento del modelo.

Fuente: Propia

3.3.4.1. Entrenamiento del Modelo

Como paso previo al entrenamiento, hay que tener en cuenta que necesitamos tener listo el

Training Dataset, este debe estar compuesto de un vector de características de la misma dimensión para todas las muestras, y cada muestra debe estar correctamente etiquetada. Las etiquetas que se escogieron para el presente trabajo son dos: “Avalanchas” y “No-avalanchas”, dentro del Dataset estos serían datos de tipo “string” o “cadena de caracteres”, y podrían mantenerse de esa forma, pero se recomienda utilizar valores numéricos, esto por si posteriormente se desea hacer alguna operación con las etiquetas. Por lo explicado, procedemos a cambiar las etiquetas por los números enteros 1 y 0; 1 como la etiqueta de evento “positivo” o “avalancha” y 0 como evento “negativo” o “no-avalancha”. De esta forma, el *Training Dataset* debería tener una estructura parecida al la figura 52.

N° Muestra	Vector de características de las muestras					Etiqueta
1	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	...	$A_{1,n}$	1
2	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$	...	$A_{2,n}$	1
3	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	...	$A_{3,n}$	1
4	$A_{4,1}$	$A_{4,2}$	$A_{4,3}$	...	$A_{4,n}$	1
5	$A_{5,1}$	$A_{5,2}$	$A_{5,3}$	...	$A_{5,n}$	1
...	...	...	...	...	...	...
96	$A_{96,1}$	$A_{96,2}$	$A_{96,3}$	...	$A_{96,n}$	0
97	$A_{97,1}$	$A_{97,2}$	$A_{97,3}$	...	$A_{97,n}$	0
98	$A_{98,1}$	$A_{98,2}$	$A_{98,3}$	...	$A_{98,n}$	0
99	$A_{99,1}$	$A_{99,2}$	$A_{99,3}$	...	$A_{99,n}$	0
100	$A_{100,1}$	$A_{100,2}$	$A_{100,3}$	...	$A_{100,n}$	0
...	...	...	...	...	...	...

Figura 52: Estructura para el *Training Dataset* que se utilizará en el entrenamiento del modelo.

Fuente: Propia

Una vez se tenga listo el *Training Dataset*, se realizará el entrenamiento de dos modelos, uno con el algoritmo de “K vecinos más cercanos” (*K-Nearest-Neighbor*) y otro con el de “Maquina de Vectores de Soporte” (*Support Vector Machine* ó *Support Vector Classifier*). Los algoritmos SVM y KNN fueron descritos en la sección 2.5.4, cada uno de ellos tienen hiperparámetros propios los cuales pueden ajustarse para modificar el comportamiento de los algoritmos, y por lo tanto, modificar también la efectividad de los modelos entrenados con dichos algoritmos.

3.3.4.2. Evaluación del Modelo

El resultado de la primera etapa de este módulo es un modelo o clasificador, en esta etapa, se tendrá que hacer una evaluación de la efectividad del mismo. Existen muchas formas de evaluación de los modelos de *Machine Learning*, pero la que se utilizará para este trabajo es la validación cruzada de K-iteraciones explicada en la sección 2.4. El valor de K que elegiremos será de K=10, que es un valor muy utilizado en la literatura.

Además, se debe probar varias combinaciones de hiperparámetros para el entrenamiento con los algoritmos SVM y KNN, a fin de hacer encontrar el grupo de hiperparámetros que resulten en el modelo de *Machine Learning* más óptimo.

La salida del Módulo de Entrenamiento del Modelo es un modelo/clasificador optimizado, al haber pasado todas las instancias de evaluación y validación.

3.3.5. Módulo de Acondicionamiento de Señal para Procesamiento en Tiempo Real

Este módulo esta compuesto de 4 principales pasos: Sensado y digitalización de la señal, transmisión por la red, almacenamiento de la data en una base de datos en tiempo real y extracción de fragmentos de análisis en tiempo real. La figura 53 representa la arquitectura del módulo propuesto.

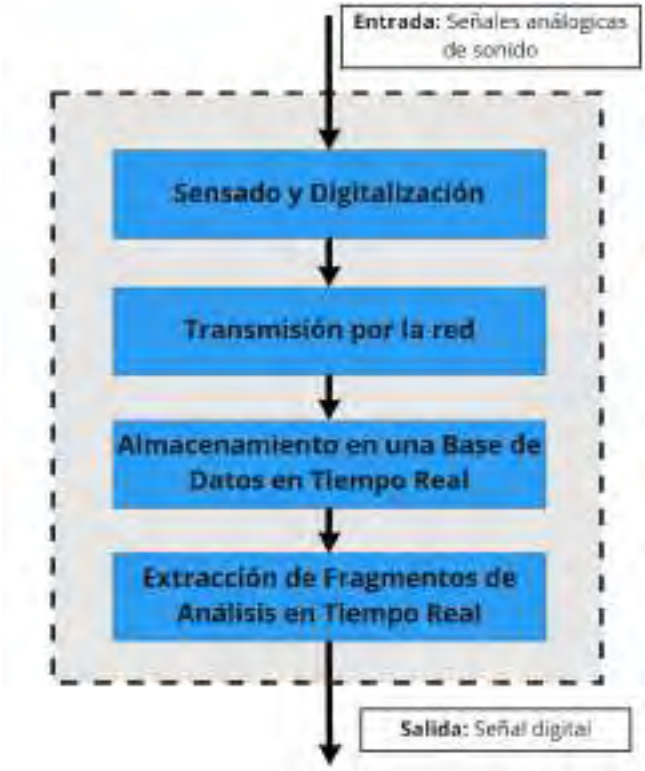

Figura 53: Arquitectura del módulo de acondicionamiento de señal para procesamiento en tiempo real.

Fuente: Propia

3.3.5.1. Sensado y Digitalización

Esta etapa del módulo se desarrolla exactamente igual a la etapa diseñada en la sección 3.3.1.1, el procedimiento es el mismo y el resultado también. El resultado de esta etapa es una señal digital de infrasonido que se almacenará en la memoria SD del *Raspberry Pi* en el formato *miniSEED*.

3.3.5.2. Transmisión por la Red

Esta etapa del módulo se desarrolla parecida a la etapa diseñada en la sección 3.3.1.2, al final de esta etapa se debe poder conectarse de forma remota al *Raspberry Pi* donde se encuentra la data de infrasonido. La única diferencia que se aprecia con la sección 3.3.1.2, es que el protocolo SFTP no es útil para los propósitos de este módulo, ya que lo que se desea no es simplemente acceder a los archivos del *Raspberry Pi*, sino que en este caso, se requiere acceder al servidor que aloja la data de infrasonido en formato *miniSEED* para poder almacenar o manipular la data en el mismo instante en que se genere, en otras palabras

el protocolo SFTP no nos es útil para el análisis en tiempo real.

El servidor que aloja la data de infrasonido es un tipo particular de servidor que funciona de forma automática como parte del software que otorga el fabricante del sensor *Raspberry Boom*, este tipo de servidor es del tipo llamado *SeedLink* y para poder hacer una transmisión de sus datos se hace por medio del protocolo del mismo nombre. El protocolo *SeedLink* es un protocolo diseñado para una robusta transmisión de datos, funciona con cualquier dispositivo que sea compatible con el protocolo TCP/IP. Este protocolo es robusto, en el sentido de que los clientes pueden desconectarse y reconectarse sin perder data, en otras palabras, la transmisión puede reanudarse siempre y cuando la data aún exista en el buffer de los servidores. Todos los paquetes de datos son archivos en formato *miniSEED* de tamaño 512 bytes. La conexión al servidor *SeedLink* se hace a través del ip asignado o a través de la URL "rs.local" en el puerto 18000.

3.3.5.3. Almacenamiento de la Data en una Base de Datos en Tiempo Real

Después de conectarse al servidor *SeedLink*, lo que procede es capturar esta data en tiempo real por medio de un cliente y luego almacenarla en una base de datos.

Antes de comenzar con el almacenamiento, cabría aclarar algunas cosas respecto al concepto de "tiempo real". De la sección anterior se sabe que la transmisión de datos por medio del protocolo *SeedLink* envía la data en paquetes de 512 bytes. Ya que se conoce el tamaño de la data para 24 horas según la documentación (15 mb) [36], entonces se puede calcular que los paquetes están compuestos de archivos que comprenden señales de entre 2 y 3 segundos de duración; esto quiere decir que la data es transmitida cada 2 o 3 segundos lo cuál es lo más cercano a tiempo real que se puede llegar con el protocolo *SeedLink*. Aunque la data no llegue de forma instantánea, sigue siendo un tiempo lo suficientemente pequeño como para considerarse dentro de lo que se conoce como "tiempo real".

Siguiendo con el procedimiento, se debe capturar la data por medio de un cliente, para esto nos apoyaremos de la librería *Obspy*, que como se explicó en la sección 3.2.2 y como se puede observar en su documentación [37], provee de un cliente que permite el acceso a servidores *SeedLink*. La librería también permite manipular o procesar cada paquete que va llegando, esto es muy importante ya que nos permitirá enviar la data a una base de datos en un servidor externo; para ello, debemos elegir y crear la base de datos que almacenará los datos de la serie de tiempo.

Para este proyecto se decidió utilizar la base de datos *influxDB*, en primer lugar por su naturaleza de base de datos no-relacional que permite un manejo de grandes cargas de información (en solo 1 minuto el sensor puede generar hasta 6000 puntos de data, donde cada punto es un registro para la base de datos), además su naturaleza simple en la estructura es ideal para series de tiempo. En segundo lugar, otra razón importante para elegir esta base de datos es que entre las bases de datos no relacionales, esta se ha especializado en aplicaciones de monitoreo de sensores, internet de las cosas, análisis en tiempo-real de series de tiempo, etc.

La forma en como *influxDB* almacena la data es parecida al tipo de dato diccionario de *Python*, el cual esta compuesto de una "clave" y un "valor", de forma análoga, en *influxDB* cada registro tiene la estructura de clave-valor, el cuál compone un "campo" o "*field*", cada "campo" esta siempre asociado una marca de tiempo, una "medición" o "*measurement*" es un básicamente una estructura de data que describe un "campo" asociado.

Lo que sigue es crear la base de datos y hacerla correr en un servidor. Según la documentación de *influxDB* [38], la base de datos utiliza el puerto de red 8086 para la comunicación cliente-servidor. Hasta este momento habíamos habilitado un cliente para la comunicación con el servidor *SeedLink* pero se requiere otro cliente que se conecte al servidor de la base de datos *influxDB*, este otro cliente se implementará con la librería *Python* "*influxdb-python*", este segundo cliente se conectará al puerto 8086 del host donde esta corriendo la base de datos *influxDB* y se podrá enviar y almacenar registros.

Un esquema general del proceso entero se ilustra en la figura 54.

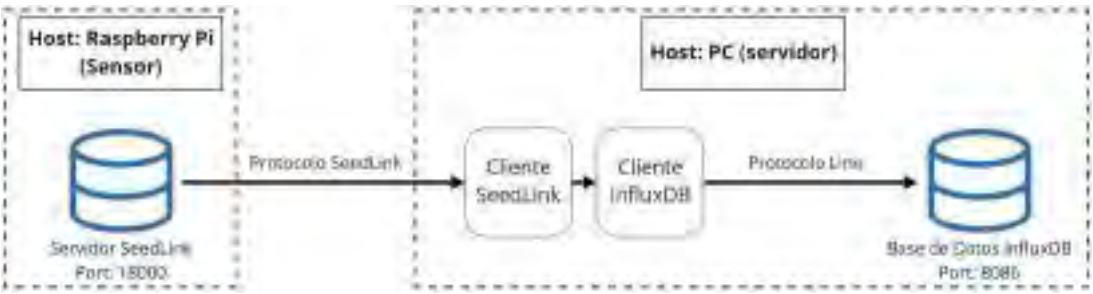

Figura 54: Esquema general del proceso de transmisión de data desde el sensor hasta la base de datos.

Fuente: Propia

3.3.5.4. Extracción de Fragmentos de Análisis en Tiempo Real

Una vez se tenga la base de datos recibiendo datos de forma continua, lo que sigue es extraer pequeños fragmentos de la señal, estos fragmentos deben estar compuestos de la data más reciente que va llegando a la base de datos, de modo que su análisis sea lo más cercano a un "análisis en tiempo real". La lógica de los fragmentos se explicó en detalle en la sección 3.2.4, en esa misma sección se definió lo siguiente sobre los fragmentos:

- Tamaño de los fragmentos: 10.24 segundos
- Tamaño de solapamiento entre fragmentos: 8.24 segundos

De la resta entre el tamaño de los fragmentos y el tamaño del solapamiento, se dedujo también, que se debería tener listo un fragmento de análisis cada 2 segundos. Por tanto, para esta etapa el objetivo es lograr extraer fragmentos de 10.24 segundos cada 2 segundos, siendo estos fragmentos compuestos de los registros más recientes que llegan a la base de datos.

Para lograr la extracción de los fragmentos, *influxDB* dispone de un lenguaje parecido a SQL que se llama *TICKscript*, al igual que con SQL, se pueden realizar consultas o *querys*, de modo que nos permita consultar de forma continua los registros más recientes de la base de datos. Con *TICKscript*, existen dos tipos de consultas continuas que podrían permitir realizar la tarea que queremos realizar: *Streaming* y *Batching*. *Streaming* permite extraer los últimos registros según va llegando, punto por punto, esta forma es ideal para tareas de monitoreo, visualización en tiempo real y para tareas de alerta inmediata, en *streaming* el procesamiento punto por punto puede poner una carga computacional muy grande en el servidor. Para tareas no tan inmediatas se utiliza el *batching*, que es básicamente la consulta de bloques de registros en intervalos regulares, el *batching* sirve para tareas de monitoreo que permiten un pequeño retraso, también es más útil cuando se trata de procesar la data, ya que el intervalo de tiempo entre consulta y consulta libera un poco la memoria del servidor para permitir el procesamiento bloque por bloque.

Según lo explicado, para el presente proyecto se decidió utilizar las consultas de tipo *batching*, ya que coinciden mejor con la lógica de los fragmentos de las que se mencionó antes. Para este proyecto, el *batching* consistirá en la consulta continua de bloques de datos de 10.24 segundos cada 2 segundos.

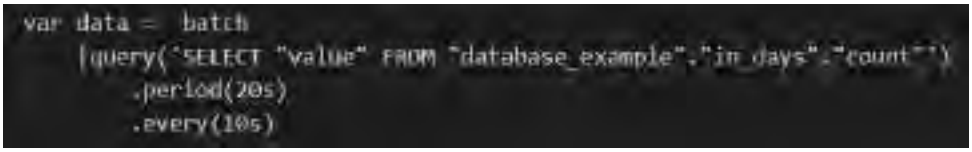

Figura 55: Ejemplo de un procedimiento de consulta de tipo *batch* en *TICKscript*, en este ejemplo se extrae bloques de 20 segundos cada 10 segundos.

Fuente: Propia

Aunque básicamente ya logramos la extracción de los registros en forma de fragmentos, esta extracción se realizó en forma de *query* o consulta, sin embargo, lo que se requiere para este proyecto es su extracción en un formato de *array* de *Python*, esto debido a que las librerías con las que se va a trabajar en el proyecto, incluyendo el entrenamiento del modelo de *Machine Learning*, están hechas con el lenguaje *Python*. De esta forma los, fragmentos de 10.24 segundos deben convertirse a *arrays* de 1024 elementos para poder seguir con el procesamiento en los posteriores módulos. Para ello, utilizaremos un motor de procesamiento de data en tiempo real llamado *Kapacitor*, este motor fue desarrollado por InfluxData, la misma empresa detrás del desarrollo de *influxDB*, esto permitirá una integración e implementación más rápida.

Kapacitor es capaz de integrarse con *influxDB* y su lenguaje *TICKscript*, su utilidad radica en que puede extraer la información de las consultas de tipo *batching* o *streaming*, para después poder llevar a cabo su procesamiento en los lenguajes GO o *Python*, también permite enviar data de regreso a *influxDB* para poder almacenar algún resultado del procesamiento si se desea. Para poder realizar el procedimiento de extracción y procesamiento con *Kapacitor* se requiere los siguientes elementos: un *script* en *TICKscript* para realizar el *batching*, un *script Python* que servirá para procesar la data recibida del *batching* y la creación de una tarea o task dentro del software de *Kapacitor* para indicar que esta tarea se realizará de forma permanente mientras la data siga llegando a la base de datos. Todo este procedimiento está definido en la documentación de *Kapacitor* [39].

En resumen, el resultado de esta etapa y todo el módulo, es un fragmento de 10.24 segundos extraído de la señal cada 2 segundos en tiempo real, este fragmento debe convertirse a un *arrays* en *Python*, de forma que sea posible su posterior procesamiento.

3.3.6. Módulo de Clasificación de Eventos

Este módulo esta compuesto de 2 principales pasos: Clasificación de eventos en tiempo real y almacenamiento del resultado de clasificación en la base de datos. La figura 56 representa la arquitectura del módulo propuesto.

Figura 56: Arquitectura del módulo clasificación de eventos.

Fuente: Propia

3.3.6.1. Clasificación de Eventos en Tiempo Real

Para esta etapa del módulo, se utilizará el modelo de *Machine Learning* optimizado del módulo explicado en la sección 3.3.4.

El procedimiento es simple, se debe importar el módulo y realizar la clasificación del vector de características procedente del procesamiento de un fragmento de análisis. Dado que los fragmentos de análisis se van generando cada 2 segundos, según lo diseñado en el módulo 3.3.5, también se tendrá un resultado de clasificación cada 2 segundos. El resultado del clasificador puede tener dos resultados, “avalancha” o “no-avalancha”, estas dos etiquetas están representadas por los número enteros 1 y 0 respectivamente, ya que esa es la equivalencia numérica que se les asignó en el entrenamiento (ver sección 3.3.4.1).

Recordar también, que para este módulo, no solo es importante el resultado de la clasificación, sino también su marca de tiempo, es decir, a que hora exacta se realizó esta clasificación. El valor de la marca de tiempo de la clasificación se debe considerar igual a la

marca de tiempo del final del fragmento de análisis.

3.3.6.2. Almacenamiento del Resultado de Clasificación en la Base de Datos

Después de obtener el resultado de la clasificación, la etapa final del módulo consiste en almacenar este resultado junto a su marca de tiempo en la base de datos. La clasificación se almacenará en su forma numérica (1 ó 0) y la base de datos será la misma que en la sección 3.3.5.3, *influxDB*.

Para poder realizar la escritura en la base de datos, se hará uso de *Kapacitor*, que según la documentación [39] permite el envío de datos a *influxDB*, a partir de aquí se debe utilizar el lenguaje *TICKscript* para almacenar la data enviada desde *Kapacitor* en la base de datos, todo este proceso se realizará cada 2 segundos según se tenga un nuevo fragmento y un resultado de clasificación.

Es importante añadir, que todo el procedimiento explicado en este módulo, se debe realizar como parte del procesamiento del fragmento que inicia en el módulo de Acondicionamiento de Señal para Procesamiento en Tiempo Real, pasa por los módulos de Pre-procesamiento, Extracción de Características y por este módulo, para finalmente terminar con la escritura del resultado de clasificación en la base de datos.

3.3.7. Módulo de Visualización del Resultado

Este módulo esta compuesto solamente por un paso: el Visualizador de señales. La figura 57 representa la arquitectura del módulo propuesto:

Figura 57: Arquitectura del módulo de visualización del resultado.

Fuente: Propia

3.3.7.1. Visualizador de Señales

El último módulo se compone de un visualizador de señales, en este módulo debemos graficar la señal de infrasonido y el resultado de la clasificación, esto a partir de la data almacenada en la base de datos *influxDB*.

Para ello nos apoyaremos de un software libre de visualización llamado Grafana. Después de instalar y ejecutar Grafana, este corre en el servidor local en el puerto 3000. Según su documentación [40], su configuración se realiza a través de su interfaz visual al ingresar mediante el navegador a la URL "localhost:3000".

La conexión con la base de datos *influxDB* es el primer paso de la configuración, el siguiente paso consiste en un simple llenado de información con respecto a la base de datos: Nombre, contraseña, puerto, señal de la data almacenada que se va a graficar, etc. Toda esta información se conoce de los anteriores módulos donde se diseñó el almacenamiento en la base de datos. El último paso es la visualización, esta funciona como una *query* continua, se debe elegir cada cuanto tiempo Grafana consultará la base de datos y se debe posicionar la gráfica en un *Dashboard*. Se accede al *Dashboard* a través de la URL "localhost:3000" al igual que la configuración. En la figura 58 se observa el ejemplo de un *Dashboard* en Grafana.

Figura 58: Ejemplo del *Dashboard* con varias gráficas e indicadores. Todos estos y más tipos de gráficas se pueden configurar en Grafana.

Fuente: Propia

Las señales que configuraremos para visualizar en Grafana serán dos: La señal de infrasonido y el resultado de la clasificación que se encuentra almacenado en la base de datos con solo valores de unos y ceros.

IV. Implementación

En este capítulo se detalla la implementación de los módulos del sistema propuesto siguiendo las indicaciones que fueron detalladas en el capítulo del diseño del sistema.

4.1. Software Utilizado

Una parte muy importante del sistema propuesto fue implementado con el lenguaje de programación *Python* (versión 3.10.5). Las librerías de *Python* utilizadas en el desarrollo del presente trabajo se muestran en la tabla 3.

Librería	Versión	Descripción
obspy	1.3.0	Provee de funciones, clientes para acceso a Data Centers y rutinas de procesamiento de señal que permite la manipulación de series de tiempo sismológicas o infrasónicas (formato miniSEED).
pandas	1.4.3	Provee herramientas de manejo y análisis de estructuras de datos. Es altamente eficiente y de fácil uso.
librosa	0.9.1	Provee de herramientas para análisis y procesamiento de señales de musica y audio
scipy	1.8.1	Provee de algoritmos y herramientas matemáticas para aplicaciones de ciencia e ingeniería. Incluye módulos para estadística, optimización, integración, algebra lineal, transformadas de Fourier, procesamiento de señales, etc.
numpy	1.22.4	Provee de herramientas de soporte para creación de objetos de tipo array multidimensionales (como vectores y matrices). También provee de rutinas para operaciones con arrays que incluyen operaciones matemáticas, lógicas, estadísticas, etc.
tsfel	0.1.4	Provee de algoritmos para ayudar en tareas de extracción de características de series de tiempo. Incluye características en los dominios estadístico, temporal y espectral.
noisereduce	2.0.1	Algoritmo para reducción de ruido en señales digitales del habla, bioacustica, señales fisiologicas, audio, etc. El algoritmo esta basado en el metodo de "puerta espectral" o "Spectral Gating".
scikit-learn	1.1.1	Provee de simples y eficientes herramientas para análisis de predicción de data, su uso más extendido es en la aplicación de Machine Learning.
sl2influxdb	0.0.0	Cliente de datos para servidores SeedLink que consulta y almacena registros de data en una base de datos influxDB.

Tabla 3: Librerías de *Python* utilizadas en la implementación del Sistema.

Fuente: Propia

Adicionalmente se utilizaron diversos softwares de uso libre para la implementación de algunos módulos, y otros para el apoyo en el análisis de data. Los softwares utilizados se

observan en la tabla 4.

Software	*Descripción*
Swarm	Software para análisis y visualización de señales de tiempo sismológicas y de infrasonido (formato miniSEED).
InfluxDB	Base de datos no-relacional de código abierto para series de tiempo. Especializado para aplicaciones que requieran almacenamiento y consulta eficiente de altas cargas de información como en análisis de señales en tiempo real, monitoreo de sensores, Internet de las Cosas, etc.
Kapacitor	Motor de procesamiento de datos en tiempo real, permite el análisis y procesamiento de señales con lenguajes de programación como Python y GO.
Grafana	Software para visualización de series de tiempo. Permite el monitoreo de señales y métricas en tiempo real mediante gráficas y dashboards.
PuTTY	Es un terminal emulador que permite la conexión a un computador remoto mediante una terminal. Los protocolos de comunicación que soporta PuTTY incluyen SSH, SCP, Telnet, etc.
WinSCP	Cliente SFTP que permite la transferencia de archivos de un computador remoto mediante una interfaz gráfica. Emplea el protocolo SSH para la comunicación.

Tabla 4: Softwares utilizados para el desarrollo del Sistema.

Fuente: Propia

4.2. Detalles de Implementación

4.2.1. Implementación del módulo de Recolección de Datos

4.2.1.1. Implementación de etapa: Sensado y Digitalización

Como se explicó en la etapa de diseño, los procesos de sensado y digitalización de la señal suceden en un circuito embebido como parte del sensor *Raspberry Boom*, por ello, la etapa de sensado consiste enteramente en la correcta configuración, instalación y ubicación del sensor.

Al adquirir el sensor de infrasonido *Raspberry Boom*, según el fabricante [36], se deben realizar algunos pasos previos para que este pueda funcionar correctamente. El primer paso es conectar el sensor a un *router* que tenga acceso a internet, esto se debe realizar por medio de un cable *ethernet* que se conecte al puerto de red del *Raspberry Pi* donde esta acoplado el sensor de infrasonido. Luego, el *Raspberry Pi* se debe conectar a internet y realizar automáticamente la actualización del software del sensor. Para comprobar que el software y el sensor están funcionando correctamente podemos ingresar a cualquier computadora que se encuentre conectada a la misma red de internet del *router* e ingresar a

la URL “rs.local” a través de un navegador web. De esta forma, accederemos a la interfaz gráfica del sensor *Raspberry Boom* (ver figura 59). En la página inicial de la interfaz nos muestra información del *Raspberry Pi* y del sensor, en la parte donde indica “Productor de datos” podemos comprobar si el sensor esta recolectando correctamente data de infrasonido (debe encontrarse en “activado”).

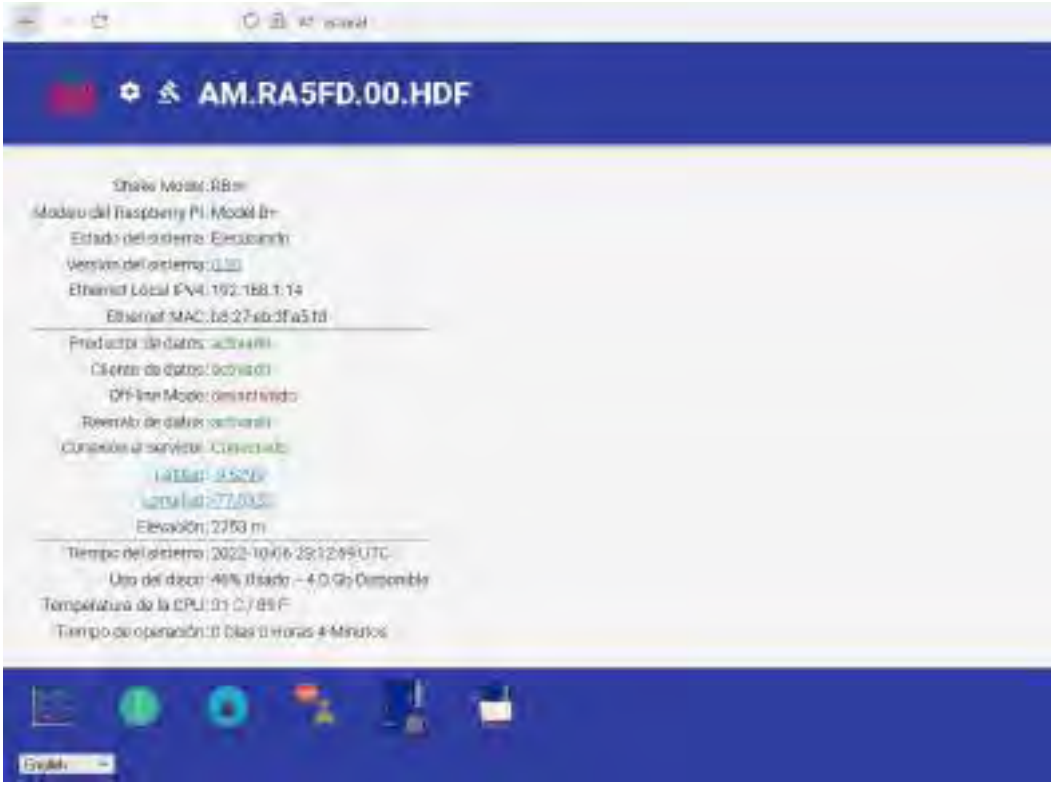

Figura 59: Página inicial de la interfaz gráfica del sensor *Raspberry Boom*.

Fuente: Propia

Luego de comprobar que el sensor esta recolectando la data debemos configurar una ip estática, esto a fin de que podamos conectarnos directamente al *Raspberry Pi*, sin importar si tenemos internet o no, o si conectamos el sensor a otra red. Para ello ingresamos a la sección de configuración en la interfaz, luego nos ubicamos en la sección red y habilitamos la opción de ip estática, la cuál será 192.168.1.14. En la figura 60 se observa la sección de la interfaz donde se realiza la configuración de red, para poder guardar cualquier cambio se debe presionar “Guardar y Reboot”.

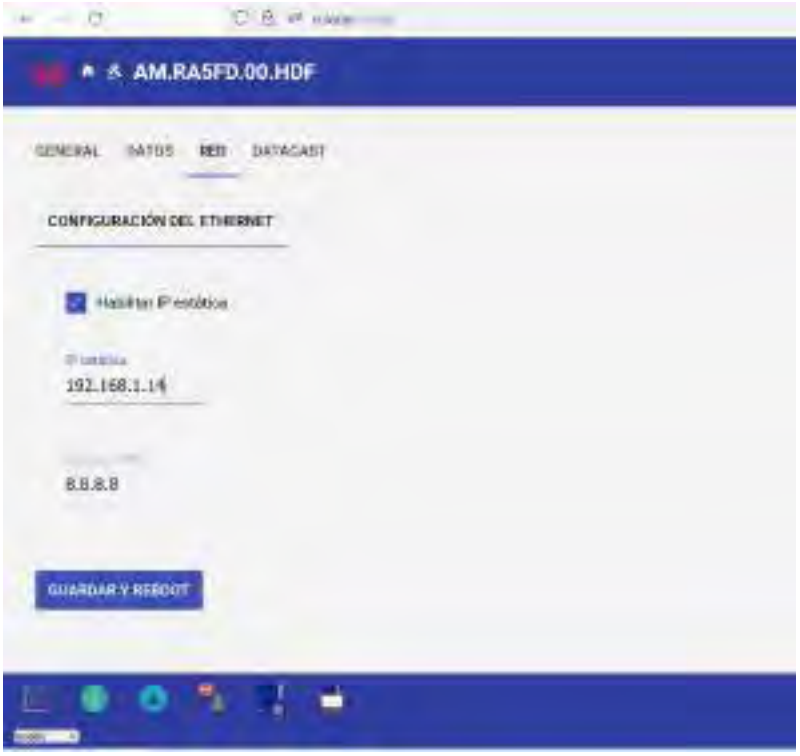

Figura 60: Página de la interfaz para configuración de red.

Fuente: Propia

Aunque el sensor ya esta listo para ser llevado a campo a ser instalado, aún queda un paso previo, el cuál esta relacionado a la atenuación de ruido para tomar mejores mediciones, para este objetivo se elaboró el *windshield* que se había diseñado en la sección 3.2.3. Este *windshield* consiste en una caja de madera hecha de madera balsa, la cuál se construyó con las siguientes dimensiones: una base cuyo tamaño interno es 13cm x 16cm, con una altura de 8.5cm y espesor de las paredes de 1.27 cm. En la figura 61 se observa la caja de madera, la cuál fue pintada con una capa de barniz para protegerla del sol y la lluvia.

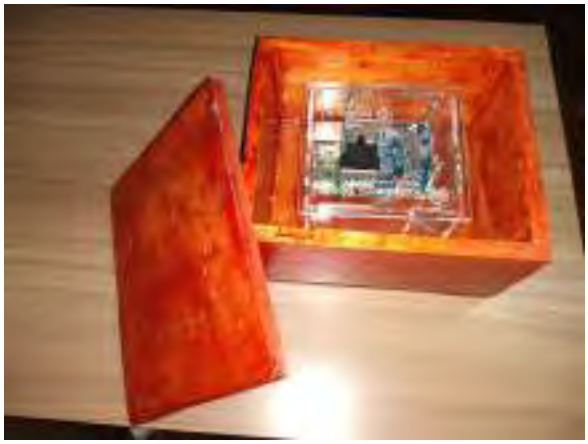

Figura 61: Caja de madera balsa, en cuyo interior se encuentra el sensor de infrasonido Raspberry Boom acoplado a un Raspberry Pi.

Fuente: Propia

Finalmente, en lo que respecta a la instalación, se ubicó el sensor cerca a donde se encuentra la antena que proporciona internet WiFi en la laguna Palcacocha, esto debido a que se requería de un lugar con acceso a energía y en esa parte existe un sistema de paneles solares que entregan una alimentación de 5v a través de un puerto USB. Además, esta ubicación cumple con la distancia mínima a la que debe estar el sensor de infrasonido de los nevados, ya que se encuentra a menos de 3 Km del lugar donde suceden las avalanchas.

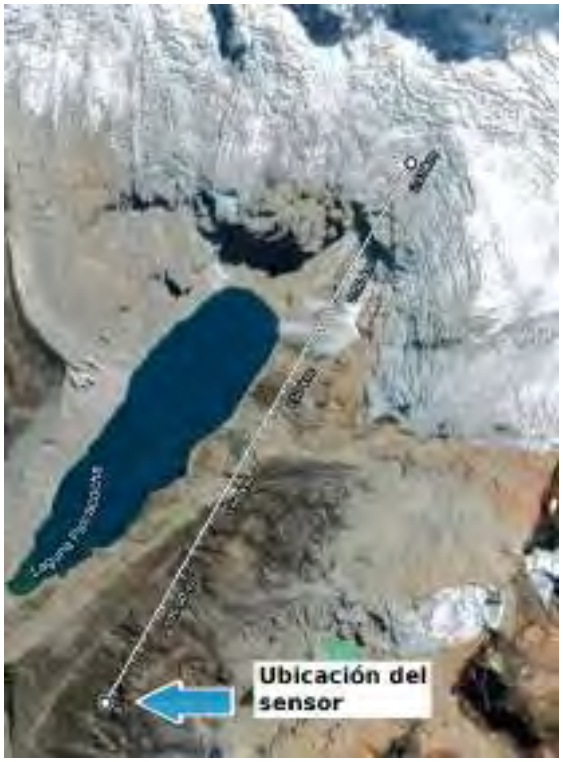

Figura 62: Ubicación del sensor y distancia hacia la zona de avalanchas.

Fuente: Propia

Figura 63: Lugar de instalación del sensor con vista al nevado y a la antena que proporciona WiFi a la laguna.

Fuente: Propia

4.2.1.2. Implementación de etapa: Transmisión por la Red

Después de la instalación del sensor, el paso siguiente es lograr acceder a este de forma remota. Por ello, se procedió a implementar esta conexión de red según lo diseñado en la sección 3.3.1.2. Como se explicó en esa sección, la conexión a la red se debe realizar a través de WiFi por medio de un módulo externo, sin embargo, antes de poder conectar este módulo al *Raspberry Pi*, corresponde realizar algunas configuraciones que se desarrollará a continuación.

En primer lugar, gracias a la ip estática que se configuró en la etapa anterior (192.168.1.14), es posible conectarse al *Raspberry Pi* sin necesidad de internet. Para conectarse con la ip estática, lo único que se necesita es una laptop y un cable *ethernet*. Se conecta el cable *ethernet* al Rapsberry Pi y a la laptop, se configura la ip del *ethernet* en la laptop de forma que se encuentre en la misma red que la ip estática en el *Raspberry Pi*, es decir, que los dos primeros grupos de números de la ip coincidan (192.168.XX.XX). De esta forma ya se puede acceder a la interfaz del sensor colocando la ip estática en lugar de "rs.local" en un navegador web.

Para poder configurar la conexión a la red, se requirió de algunas configuraciones más avanzadas que no se pueden realizar desde la interfaz del sensor, por lo que se accedió directamente a la consola terminal del *Raspberry Pi* a tráves del sofwtare PuTTY. Para ello, es necesario conectarse por SSH al ip del Rapsberry Pi (la ip estática) para poder tener acceso a la terminal del *Raspberry Pi* como se observa en la figura 64.

Figura 64: Acceso a la terminal del *Raspberry Pi* a tráves de PuTTy.

Fuente: Propia

Al haber conseguido el acceso a la terminal del *Raspberry Pi*, se procedió a realizar las configuraciones según la documentación del sensor, primero se debe comprobar que la interfaz WiFi esta activada, para ello se ingresa al archivo "interfaces" que se encuentra en la carpeta "/etc/network", y se verifica que las siguientes líneas están en dicho archivo:

```
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

Luego se procedió a editar el archivo "wpa_supplicant.conf" que se encuentra en la carpeta "/etc/wpa_supplicant" para poder configurar a que red de WiFi que se debe conectar el módulo WiFi, aquí se coloca el nombre de dicha red WiFi (ssid) y su contraseña (psk). Las siguientes líneas configurando lo mencionado se añadieron al final del archivo:

```
network={
        ssid="INAIGEM"
        psk="1n41g3m"
        proto=RSN
        key_mgmt=WPA-PSK
        pairwise=CCMP
        auth_alg=OPEN
        }
```

Para que se apliquen los cambios se reinició el *Raspberry Pi* con la instrucción sudo reboot.

Finalmente se debe conectar el módulo externo WiFi, tal como se sugirió en la etapa de diseño, primero se conectó el extensor hub USB al *Raspberry Pi* y luego el módulo WiFi a uno de los puertos USB de este hub, el cable del extensor es lo suficientemente largo como para que la antena del módulo externo tenga línea de vista con la antena que provee WiFi en la laguna. En la figura 65 se observa una foto de la instalación con el módulo externo de WiFi.

Figura 65: Instalación del sensor con el módulo externo de WiFi.

Fuente: Propia

Tener en cuenta que el módulo WiFi instalado debe ser compatible con el *Raspberry Pi*, en este caso el módulo que se instaló es de la marca REALTEK modelo RTL8188EU 802.11n, el cuál se comprobó que si es compatible. Para comprobar que el módulo se conectó sin problemas a la red WiFi se debe hacer ping a la ip del DNS de google (8.8.8.8), así se puede comprobar que el *Raspberry Pi* tuvo una conexión exitosa a la internet.

Habiendo instalado correctamente el módulo externo de WiFi y configurado la conexión del *Raspberry Pi* a la red, lo único que quedaría en esta etapa es la descarga de la data del sensor de infrasonido. Para esto se hizo del uso del software WinSCP, que permite la conexión a un dispositivo remoto mediante SSH al igual que PuTTY pero también permite la transferencia de archivos a tráves del protocolo SFTP. En la figura 66 se observa la interfaz con los campos completados para proceder con la conexión al *Raspberry Pi*.

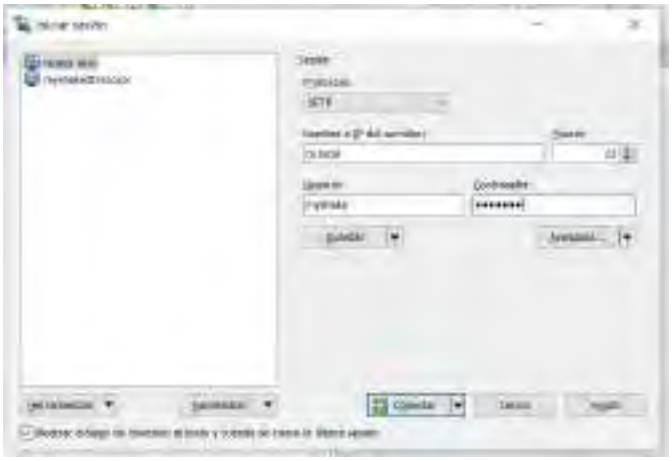

Figura 66: Interfaz gráfica de conexión WinSCP.

Fuente: Propia

Luego se debe ubicar la carpeta donde se encuentran los archivos *miniSEED* que albergan la información de las señales de infrasonido y finalmente se procede a descargarlos a la PC desde donde se hizo la conexión. La carpeta donde se ubican los archivos es "/opt/data/archive", en particular, dentro de algunas carpetas interiores que ordenan la data por estación y por año. En la figura 67 se puede observar los archivos *miniSEED* en la interfaz gráfica de WinSCP.

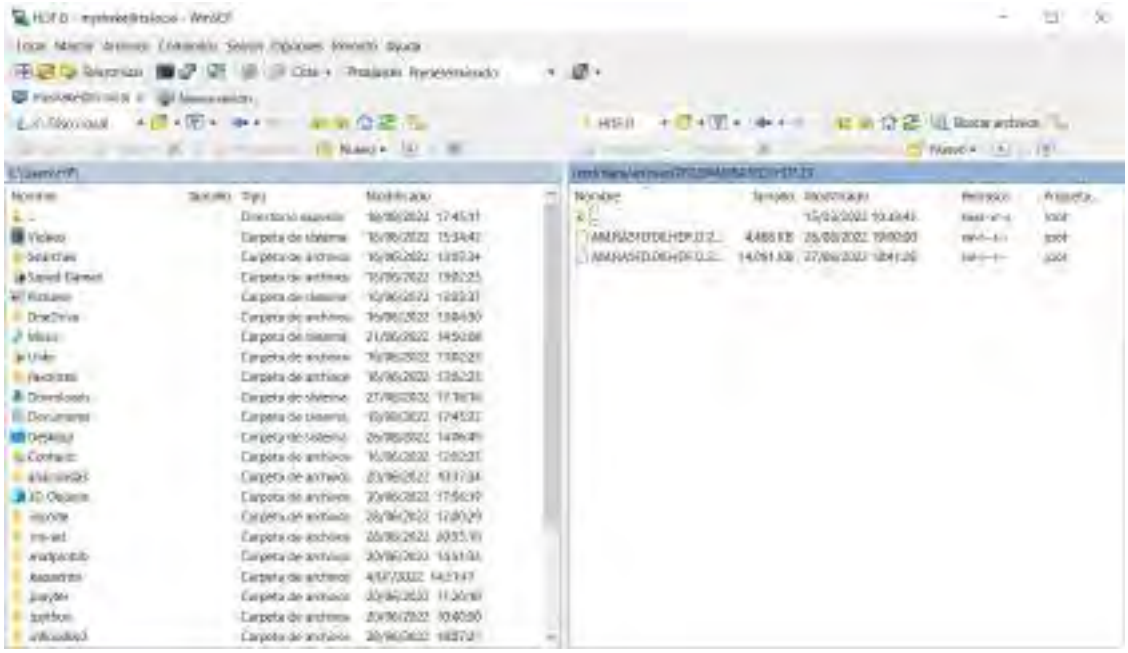

Figura 67: Interfaz gráfica para descarga de datos con WinSCP.

Fuente: Propia

4.2.1.3. Implementación de etapa: Generación del Training Dataset

En la última etapa del módulo de recolección de datos, se realizó la generación del *Training Dataset*. Como se explicó en la sección 3.3.1.3, se hizo un registro objetivo de avalanchas con la ayuda de la cámara que es parte del sistema de monitoreo que se encuentra instalada en la laguna Palacocha.

El registro de avalanchas esta compuesto de 74 avalanchas, de las cuales 2 fueron registradas el día 07 de diciembre del 2021, 1 fue registrada el día 22 de abril del 2022 y las demás 71 fueron registradas entre los días 6 y 12 de mayo del 2022. La mayoría de estos registros se obtuvieron con ayuda de la cámara pero algunas también se obtuvieron de forma presencial.

El registro de ruidos esta compuesto de 115 muestras, las cuales se obtuvo de forma presencial en la laguna entre los días 7 y 12 de mayo del 2022, se hizo así para asegurarse de registrar ruidos con gran diversidad de procedencias y que sean frecuentes en la laguna, se registraron en su mayor parte ruidos de viento, cuya forma de señal de infrasonido era muy parecida al de una avalancha, también se registraron ruidos de derrumbes de rocas no relacionadas a las avalanchas y de aviones que pasaban a una distancia no tan cercana de la laguna pero que se observó generaban señales de infrasonido en el sensor.

Cabe añadir que, según el diseño, las muestras de avalanchas del *Training Dataset* solo pueden ser de 10.24 segundos, sin embargo, se observó que muchas avalanchas duraban mucho más que 10 segundos, por lo que, para aprovechar lo máximo de estas avalanchas se les separó en varias muestras, por ejemplo, la señal de infrasonido de la avalancha de la figura 68 tiene una duración de aproximadamente 20 segundos, y ya que supera ampliamente el tope de 10.24 segundos, se le dividió en tres muestras de 10.24 segundos, los cuales comienzan en las siguientes horas: 15:43:19, 15:43:22 y 15:43:25 (Hora UTC).

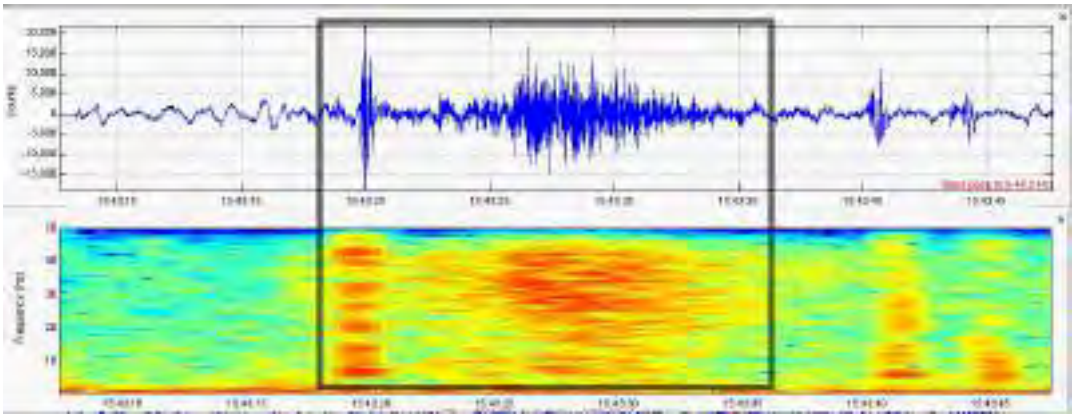

Figura 68: Señal de infrasonido de una avalancha (serie de tiempo y frecuencia).

Fuente: Propia

Siguiendo el procedimiento del anterior párrafo se pudo incrementar el número de muestras de avalanchas a 105. Logrando de esta forma un número similar en el número de muestras de avalanchas y ruidos.

Teniendo listo el registro de avalanchas y ruidos, se procedió a su almacenamiento en forma de csv, para esto primero se obtuvo la información correspondiente de las muestras de su correspondiente archivo *miniSEED* en forma de *array* de *Python*. La implementación de la función que obtiene el mencionado *array* se muestra en el código 1.

```python
from obspy import read
from obspy.core import UTCDateTime

# La entrada de fecha_inicio_señal debe ser de tipo string (texto) con el
# siguiente formato: "2022-05-12T15:43:19", la duración_señal debe ser un
# número de tipo entero y su cantidad debe representarse en segundos.
# La entrada de archivo_miniSEED es el nombre del archivo miniSEED que
# contiene la señal, es decir, debe ser un archivo del día que contiene a la señal
# que se quiere obtener.

def obtener_señal_de_archivo_miniSEED(fecha_inicio_señal, duracion_señal,
                                      archivo_miniSEED):
    """Obtiene una señal de infrasonido en forma de array numpy a partir
    de un archivo miniSEED que lo contiene"""

    #Convertir la fecha de string a formato UTCDateTime
    fecha_inicio_señal = UTCDateTime(fecha_inicio_señal)

    # Para poder cargar el archivo miniseed este debe estar en la misma carpeta
    señal = read(archivo_miniSEED, starttime=fecha_inicio_señal,
                 endtime=fecha_inicio_señal+duracion_señal)
    # La función retorna el array numpy de la señal indicada, según la fecha de
    # inicio y su duración indicada
    return señal[0].data
```

Código 1: Obtención de una señal en un *array* a partir de archivo *miniSEED*.

Luego de obtener el *array*, se procedió a su almacenamiento como archivo csv, para ello se implementó una función que almacena una señal de formato *array* en un archivo csv, también se implemento la función de carga por si en un futuro se requiere recuperar esta señal del archivo csv de vuelta a un *array* de *Python*. Dicha implementación se muestra en el código 2.

```python
import numpy as np

# El path es la carpeta donde el archivo csv será almacenado (para la primera función)
# o de donde será cargado (para la segunda función).
# El nombre del archivo csv debe llevar la extension csv, por ejemplo:
# "ejemplo_archivo.csv", esto debe ser así para ambas funciones.

def guardar_array_en_csv(path, nombre_archivo_csv, array_señal):
    np.savetxt(path+nombre_archivo_csv, array_señal, delimiter=',')

def cargar_csv_en_array(path, nombre_archivo_csv):
    array_señal = np.genfromtxt(path+nombre_archivo_csv,
                                delimiter=',')
    # Esta función retorna un array numpy a partir de un archivo csv.
    return array_señal
```

Código 2: Guardado de *array* en archivo CSV y cargado de archivo CSV en un *array*.

En conclusión, el *Training Dataset* que se utilizará en la implementación del modelo y el sistema de detección tiene las siguientes características:

- Cantidad de muestras de Avalanchas: 105
- Cantidad de muestras de No-avalanchas: 115
- Duración de todas las muestras: 10.24 segundos
- Formato de almacenamiento: csv
- Número total de muestras/archivos csv: 220

4.2.2. Implementación del módulo de Pre-procesamiento

Como se describió en la sección 3.3.2, este módulo esta compuesto de tres etapas de filtrado que se componen de los siguientes filtros: Filtro Pasabandas (1.8 - 45 Hz), Filtro Rechazabanda (25 - 30 Hz) y el Filtro de Reducción de Ruido. El módulo requeriría normalmente como entrada solo la señal a filtrar (en formato *array*) y su tasa de muestreo como número entero (100 Hz), sin embargo, debido a que el filtro de reducción de ruido requiere de una muestra de una señal de ruido, también se necesita pasarle dicha señal como entrada del módulo. La salida del módulo es un *array* que tiene la misma longitud que la de la señal de entrada.

La función del código 3 muestra la implementación del módulo.

```
from scipy import signal
import noisereduce as nr

def modulo_pre_procesamiento(señal, señal_ruido, tasa_muestreo = 100):
    '''Módulo de pre-procesamiento de señales digitales,.
    Hace un filtrado de bandas no deseadas y de ruido'''

    #Etapa del filtro Butterworth pasabanda.
    filtro_pasabanda = signal.butter(6, [1.8,45], 'bandpass',
                                     fs=tasa_muestreo, output='sos')
    señal_filtrada_1 = signal.sosfilt(filtro_pasabanda, señal)

    #Etapa del filtro Butterworth rechazabanda.
    filtro_rechazabanda = signal.butter(6, [25,30], 'bandstop',
                                        fs=tasa_muestreo, output='sos')
    señal_filtrada_2 = signal.sosfilt(filtro_rechazabanda, señal_filtrada_1)

    #Etapa del filtro de reducción de ruido.
    señal_filtrada_3 = nr.reduce_noise(y = señal_filtrada_2, n_fft=128,
                                       hop_length=64, sr=tasa_muestreo,
                                       n_std_thresh_stationary=1,
                                       stationary=True,
                                       time_mask_smooth_ms=5000,
                                       freq_mask_smooth_hz=1.8,
                                       y_noise = señal_ruido)
    return señal_filtrada_3
```

Código 3: Módulo de pre-procesamiento.

En lo que respecta a la muestra de la señal de ruido que se requiere pasarle como entrada al módulo, primero se procedió a elegir que muestra pasarle, según el diseño, esta debe ser una muestra de silencio de 20 segundos. Se eligió una muestra de 20 segundos, de un archivo *miniSEED* obtenido por el sensor, correspondiente al día 09 de mayo del 2022 a horas 01:48:41 hasta las 01:49:01 (Hora UTC), en la figura 69 se observa la muestra de "silencio" elegida.

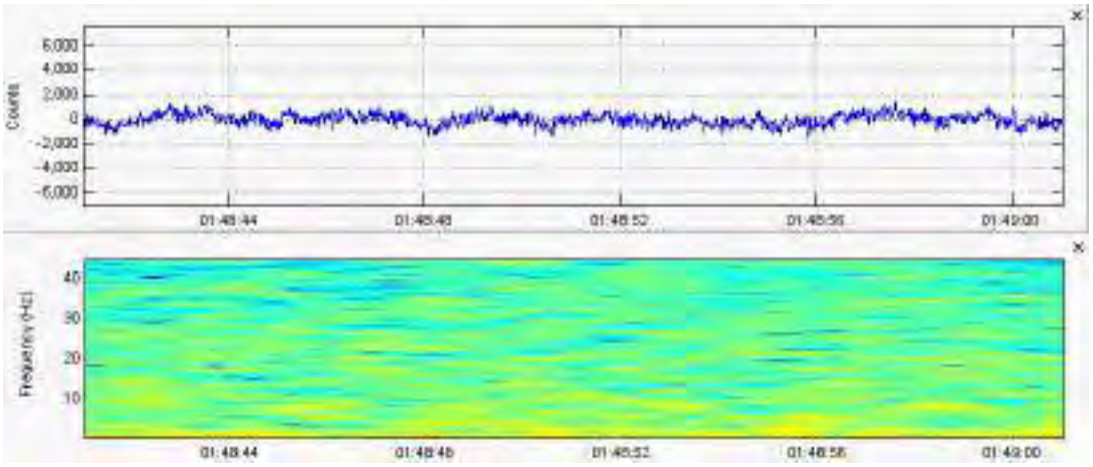

Figura 69: Señal de infrasonido de 20 segundos correspondiente a muestra de “silencio”.

Fuente: Propia

Para poder transformar la señal de ruido a un formato *array*, se utilizó la función implementada en el código 1. Así, se le pasó como entrada de la función el nombre del archivo del día entero donde se encuentra la muestra y luego se extrajo la señal indicándole la fecha de inicio y su duración. Tener en cuenta que esta señal de ruido solo se extrae una vez y se aplica como señal de entrada en el módulo de pre-procesamiento en todos los casos.

4.2.3. Implementación del módulo de Extracción de Características

Como se explicó en la sección 3.3.3, el resultado de este módulo debe ser un vector que este compuesto por el patrón del características, la búsqueda del patrón de características requiere una serie de pruebas para examinar el nivel de correlación entre las características (*scattering*). Aunque las pruebas y los resultados se realizarán en el capítulo siguiente, es parte de esta sección la implementación de la extracción de distintas características descritas en la sección de diseño.

Las características asociadas a la “Relación de Energía entre Bandas” se implementarán con ayuda de la librería librosa, todas las demás características se implementarán con la ayuda de la librería TSFEL [41].Como el grupo de características más numeroso fue extraído con la librería TSFEL, se detallará todo lo relacionado a esta librería primero.

Según la documentación de la librería TSFEL, para poder extraer las características de una serie de tiempo se debe realizar a través de un archivo JSON, el cuál tiene una

estructura definida según la misma documentación, de esta forma, antes de comenzar con la extracción se definió dicho archivo JSON y se guardo con el nombre “features.JSON”. Este JSON lleva incluidas las siguientes características: Centroide Espectral, Kurtosis Espectral, Envergadura Espectral, Tasa de Cruce por Cero, Puntos de Giro Positivos Espectrales, Roll-off Espectral, Asimetría Espectral y Pendiente Espectral. El contenido del JSON puede observarse en el anexo 03.

Lo siguiente fue extraer la característica de “Relación de Energía entre Bandas” o “*Band Energy Ratio*” (BER). Como se vio en la sección 3.3.3, esta relación puede derivarse en muchas características, dependiendo cuantas relaciones y entre cuales bandas se desee obtener la relación, la función que se utilizará para cada una de estas relaciones esta definida por una sola función, la cuál se implementó basado en la fórmula 15. La implementación se hizo en dos partes: la primera función corresponde a una simple herramienta de transformación de la frecuencia en Hz a su contra parte en bins, se requiere está función para la segunda parte que es la extracción de la característica BER para cada ventana. La implementación se puede observar en el código 4.

```python
import numpy as np

def calcular_bin_frequencia(frecuencia_hz, espectrograma, tasa_muestreo):
    '''Función que transforma una frecuencia en Hz en su respectivo bin de
    frecuencia. Requiere como entrada el espectrograma de la señal que se
    quiere analizar'''

    rango_total_frecuencia=tasa_muestreo/2

    # Se calcula el delta de frecuencia de cada bin del espectrograma, es
    # decir la distancia entre un bin y otro (en hz).
    frequency_delta_por_bin=rango_total_frecuencia/ espectrograma.shape[0]

    # Se calcula el bin de frecuencia y se retorna el mismo como resultado de la
    # función.
    frecuencia_bin=np.floor(frecuencia_hz/frequency_delta_por_bin)
    return int(frecuencia_bin)

def calcular_band_energy_ratio(espectrograma, inicio_band_1, fin_band_1,
                               inicio_band_2, fin_band_2,tasa_muestreo):
    '''Función que calcula la "Relación de Energía entre Bandas" o "Band Energy
    Ratio". Requiere como entrada el espectrograma de la señal que se quiere
    analizar'''

    # Calcular el bin de cada una de las frecuencias que se le pasaron a la
    # función como parámetros de entrada.
    frecuencia_bin_inicio_band_1=calcular_bin_frequencia(espectrograma,
                                                         inicio_band_1,
                                                         tasa_muestreo)
    frecuencia_bin_fin_band_1=calcular_bin_frequencia(espectrograma,
                                                      fin_band_1,
                                                      tasa_muestreo)
    frecuencia_bin_inicio_band_2=calcular_bin_frequencia(espectrograma,
                                                         inicio_band_2,
                                                         tasa_muestreo)
    frecuencia_bin_fin_band_2=calcular_bin_frequencia(espectrograma,
                                                      fin_band_2,
                                                      tasa_muestreo)
    # Calcular potencia del espectrograma
    power_spec=np.abs(espectrograma)**2
    power_spec=power_spec.T

    # Calcular la band energy ratio para cada ventana/frame del espectrograma
    # El resultado es un array con los valores de BER para cada frame,
    # la longitud del array esta definido por el número de frames/ventanas
    # totales de la señal.
    band_energy_ratio = []
    for frequencies_in_frame in power_spec:
        sum_power_band1_frequencies=np.sum(frequencies_in_frame
                                           [frecuencia_bin_inicio_band_1:
                                            frecuencia_bin_fin_band_1])
        sum_power_band2_frequencies=np.sum(frequencies_in_frame
                                           [frecuencia_bin_inicio_band_2:
                                            frecuencia_bin_fin_band_2])
        ber_frame_actual=sum_power_band1_frequencies/ sum_power_band2_frequencies
        band_energy_ratio.append(ber_frame_actual)

    # La función retorna el array numpy con los valores de la característica BER
    # por cada frame de una señal.
    return np.array(band_energy_ratio)
```

Código 4: Extracción de las características de BER.

El número de características de BER que habían sido definidas en la sección de diseño

3.3.3 eran solo 3, sin embargo, en una observación preliminar de la data se observó que la energía de las señales de ruido generalmente se concentra en la banda de 0-5 Hz, además de que las señales de avalanchas tienen componentes notables en la banda de 10-40 Hz y que pueden servir para discriminarlas del ruido, por ello, se decidió añadir 3 relaciones más. Las características de BER que se extraerán en total serán las siguientes:

- Relación de energía de la banda de 6-8 Hz en relación a la banda total (0-50 Hz).
- Relación de energía de la banda de 26-35 Hz en relación a la banda total (0-50 Hz).
- Relación de energía de la banda de 7-16 Hz en relación a la banda de 16-37 Hz.
- Relación de energía de la banda de 5-11 Hz en relación a la banda total (0-50 Hz).
- Relación de energía de la banda de 11-35 Hz en relación a la banda total (0-50 Hz).
- Relación de energía de la banda de 0-5 Hz en relación a la banda de 5-50 Hz.

Utilizando las funciones para extracción del BER y definiendo como entrada el nombre del archivo JSON se pudo finalmente implementar el módulo de extracción de características. En total el número de características iniciales que habían eran: las 8 extraídas con la librería TSFEL y las 6 de BER, siendo un total de 14. Según lo explicado, el módulo de extracción de características se implementó de la siguiente forma:

```python
from tsfel import time_series_features_extractor
from tsfel import load_json
from librosa import stft
import pandas as pd

def modulo_extraccion_caracteristicas(señal, tasa_muestreo, archivo_JSON,
                                      FRAME_LENGTH, HOP_LENGTH):
    '''Módulo de extracción de características para una señal segmentada en frames.
    Calcula un vector que es la unión de todos los vectores de características de
    cada frame'''

    # Carga del archivo JSON por su nombre, este debe encontrarse en la misma
    # carpeta del script.
    JSON_características = load_json(archivo_JSON)

    # Obtención del espectrograma de la señal, este se utilizará en la extracción
    # de las características de tipo BER.
    espectrograma=stft(señal, n_fft=FRAME_LENGTH, hop_length=HOP_LENGTH, center=False)

    # Extracción de las 8 primeras características con TSFEL, notar que en el
    # parámetro "overlap" se le pasa un porcentaje de solapamiento con respecto
    # al tamaño total del frame. El resultado es un Dataframe de pandas que en este
    # caso corresponde a una matriz (de tamaño: nº frames X 8 características).
    matriz_caracteristicas=time_series_features_extractor(JSON_características,
                                                          señal, fs=tasa_muestreo,
                                                          window_size=FRAME_LENGTH,
                                                          overlap=
                                                          (FRAME_LENGTH-HOP_LENGTH)/
                                                          FRAME_LENGTH)

    # Extracción de las 6 características de tipo BER. Cada una de estas
    # instrucciones genera un array de tamaño igual al nº de frames
    # (1 característica por cada frame).
    ber_signal_band_to_band1=calcular_band_energy_ratio(espectrograma,
                                                        6,8,0,50, tasa_muestreo)
    ber_signal_band_to_band2=calcular_band_energy_ratio(espectrograma,
                                                        26,35,0,50, tasa_muestreo)
    ber_signal_band_to_band3=calcular_band_energy_ratio(espectrograma,
                                                        7,16,16,37, tasa_muestreo)
    ber_signal_band_to_band4=calcular_band_energy_ratio(espectrograma,
                                                        5,11,0,50, tasa_muestreo)
    ber_signal_band_to_band5=calcular_band_energy_ratio(espectrograma,
                                                        11,35,0,50, tasa_muestreo)
    ber_signal_band_to_band6=calcular_band_energy_ratio(espectrograma,
                                                        0,5,5,50, tasa_muestreo)

    # Se concatena el grupo de características BER al de TSFEL utilizando
    # librería pandas. El resultado será un dataframe de pandas que corresponde
    # a una matriz (de tamaño: nº frames X 14 características).
    matriz_caracteristicas["BER band/band1"]=ber_signal_band_to_band1
    matriz_caracteristicas["BER band/band2"]=ber_signal_band_to_band2
    matriz_caracteristicas["BER band/band3"]=ber_signal_band_to_band3
    matriz_caracteristicas["BER band/band4"]=ber_signal_band_to_band4
    matriz_caracteristicas["BER band/band5"]=ber_signal_band_to_band5
    matriz_caracteristicas["BER band/band6"]=ber_signal_band_to_band6

    # El dataframe de la matriz se convierte a un array y se realiza el flattening.
    vector_caracteristicas=matriz_caracteristicas.values.flatten()

    # El resultado del módulo es un vector unidimensional (numpy array).
    # El tamaño del array esta determinado por el número de frames, cuya
    # cantidad esta determinada a su vez por el tamaño de los Frames y su
    # solapamiento (FRAME_LENGTH y HOP_LENGTH).
    return vector_caracteristicas
```

Código 5: Módulo de extracción de características.

Los parámetros de entrada FRAME_LENGTH y HOP_LENGTH que se observan en la función implementada en el código 5 están definidos según diseño como 256 y 128 respectivamente, esto da como resultado que cada fragmento de señal tendrá 7 *frames*/ventanas (ver sección 3.2.5), por lo que el vector de características resultante de este módulo debería tener 98 elementos (14 características x 7 *frames*).

Es importante mencionar también que el módulo implementado en el código 5 se utiliza en dos etapas del sistema, la primera es en la búsqueda del modelo de *Machine Learning* más óptimo, y la segunda es en la etapa de procesamiento en tiempo real, para esta última etapa ya se debe haber obtenido el patrón de características más óptimo y por tanto algunas características extraídas más arribas podrían quedar descartadas, de darse este caso, simplemente correspondería eliminar del archivo JSON las características TSFEL que se descartaron y eliminar las líneas del código 5 correspondientes a las características BER descartadas.

4.2.4. Implementación del módulo de Entrenamiento del Modelo

4.2.4.1. Implementación de etapa: Entrenamiento del modelo

Antes de comenzar habría que aclarar que la entrada de este módulo es un dataset de entrenamiento compuesto de un grupo de vectores de características y una columna que corresponde a las etiquetas o clases, esto se precisó en la sección 3.3.4.1. En la figura 52 se observa la estructura que debe tener la entrada de este módulo, en la implementación esta estructura corresponde a un *array* de dos dimensiones. Para propósitos de evaluación del modelo se convirtió el *Training Dataset*, de un *array* bidimensional (matriz) a un *Dataframe* de Pandas, la siguiente función se implementó con dicho propósito:

```
import pandas as pd

def convert_training_dataset_to_dataframe(array_train_dataset, tasa_muestreo,
                                          longitud_señal, FRAME_LENGTH, HOP_LENGTH,
                                          lista_caracteristicas):
    # Se define los headers o encabezados para las caracteristicas del Dataframe.
    header_caracteristicas=lista_caracteristicas

    num_ventanas_señal=int((((longitud_señal*tasa_muestreo)-FRAME_LENGTH)/
                            HOP_LENGTH)+1)

    # Se repite varias veces el mismo conjunto de headers, ya definidos en la
    # anterior instrucción. Esto se realiza según el número de ventanas/frames.
    header_caracteristicas=header_caracteristicas * num_ventanas_señal

    # Se coloca un header extra en la última columna que corresponde a las
    # etiquetas de ''clase''.
    header_caracteristicas.append('clase')

    # Se crea el dataframe indicadole los headers/encabezados de las demás
    # columnas a partir de la lista_caracteristicas que es entrada de la función.
    dframe_train_dataset = pd.DataFrame(array_train_dataset,
                                        columns=header_caracteristicas)

    # El resultado es el dataframe del array del training Dataset, este dataframe
    # tiene headers incluidos que ayudaran para un análisis visual mas eficiente.
    return dframe_train_dataset
```

Código 6: Conversión del *Training Dataset* en un *Dataframe* de la librería Pandas.

Esta función permite obtener un *Dataframe* a partir de un *array*, siempre y cuando se le pase por entrada la lista de todas las características disponibles, para este trabajo la lista tiene la siguiente forma:

```
lista_caracteristicas_total=['0_Spectral centroid', '0_Spectral kurtosis',
                             '0_Spectral positive turning points',
                             '0_Spectral roll-off', '0_Spectral skewness',
                             '0_Spectral slope', '0_Spectral spread',
                             '0_Zero crossing rate', 'BER 6-8/total',
                             'BER 26-35/total', 'BER 7-16/16-37',
                             'BER 5-11/total', 'BER 11-35/total', 'BER 0-5/5-50']
```

El *Training Dataset* obtenido en el módulo de recolección de Datos tiene 220 muestras, donde cada muestra esta compuesta de 7 ventanas, y a cada ventana le corresponde un vector de 14 características, además la ultima columna del Dataset corresponde a las etiquetas (clases), por lo que el *Dataframe* resultante tiene las siguientes dimensiones:

- Num. de Filas: 220 (220 muestras)
- Num. de Columnas: 99 ((14 caracteristicas x 7 ventanas) + 1 Columna de clase)

Considerando lo anteriormente explicado, se procedió con la implementación de la función de entrenamiento del modelo. Para este trabajo se implementaron dos funciones, ya

que como se explicó en el capítulo de diseño, se evaluaran dos algoritmos: SVM (*Support Vector Machine*) y KNN (*K-Nearest-Neighbor*):

```
from sklearn import svm

def entrenamiento_modelo_SVM(gamma, C, dataframe_training_dataset):

    ''' Función encargada de entrenar un modelo de Machine Learning con el
    algoritmo SVM a partir de un training dataset que se encuentra en una
    estructura de tipo Dataframe (Pandas), tiene como entrada también los
    hiperparámetros de entrenamiento'''

    # Dataframe correspondiente a los vectores de características
    # (Es el mismo dataframe del Training Dataset sin la columna "clase")
    dataframe_vectores_caracteristicas = dataframe_training_dataset.\
                                        loc[:,dataframe_training_dataset.\
                                            columns != 'clase']
    # Dataframe correspondiente a la columna de clase
    # (Corresponde a un dataframe de solo la columna "clase")
    dataframe_columna_clase=dataframe_training_dataset['clase']

    # Eleccion de hiperparametros y entrenamiento de modelo
    modelo_SVM= svm.SVC(gamma=gamma , C=C)
    modelo_SVM.fit(dataframe_vectores_caracteristicas.values,
                   dataframe_columna_clase.values)

    # esta función devuelve un objeto de tipo SVC, que es un objeto de la
    # libreria scikit-learn que representa un modelo de SVM.
    return modelo_SVM
```

Código 7: Entrenamiento de modelos de *Machine Learning* con el algoritmo SVM.

```
from sklearn import neighbors

def entrenamiento_modelo_KNN(n_neighbors, weights, metric,
                             dataframe_training_dataset):

    ''' Función encargada de entrenar un modelo de Machine Learning con el
    algoritmo KNN a partir de un training dataset que se encuentra en una
    estructura de tipo Dataframe (Pandas), tiene como entrada también los
    hiperparámetros de entrenamiento'''

    # Dataframe correspondiente a los vectores de caracteristicas
    # (Es el mismo dataframe del Training Dataset sin la columna "clase")
    dataframe_vectores_caracteristicas = dataframe_training_dataset.\
                                         loc[:,dataframe_training_dataset.\
                                             columns != 'clase']
    # Dataframe correspondiente a la columna de clase
    # (Corresponde a un dataframe de solo la columna "clase")
    dataframe_columna_clase=dataframe_training_dataset['clase']

    # Eleccion de hiperparametros y entrenamiento de modelo
    modelo_KNN= neighbors.KNeighborsClassifier(n_neighbors=n_neighbors,
                                               weights=weights,
                                               metric=metric)
    modelo_KNN.fit(dataframe_vectores_caracteristicas.values,
                   dataframe_columna_clase.values)

    # esta función devuelve un objeto de tipo KNeighborsClassifier, que
    # es un objeto de la libreria scikit-learn que representa un modelo
    # de KNN.
    return modelo_KNN
```

Código 8: Entrenamiento de modelos de *Machine Learning*, con el algoritmo KNN.

Otra herramienta importante que se requiere, es la función de exportación del modelo, si bien con la función de arriba se pudo obtener un modelo, este es un objeto que solo funciona en el contexto de un *script* de *Python*, si se quiere almacenar el modelo como un archivo, se debe implementar una función que lo exporte. Esta función y su correspondiente función para cargar el archivo se implementaron en el código 9

```python
# El módulo pickle es un módulo que se encuentra incluido por defecto en Python
import pickle

# El nombre del achivo que se pasa como entrada a ambas funciones
# debe tener el formato siguiente: "ejemplo.pkl"

def exportar_modelo (modelo, nombre_de_archivo, path):
    '''Esta función esta encargada de exportar un modelo de Machine Learning de
    python en un archivo de formato pickle (*.pkl)'''

    #exportar el archivo con un nombre y en una carpeta determinada
    pickle.dump(modelo,open(path + nombre_de_archivo, 'wb'))

def cargar_modelo (nombre_de_archivo, path):
    '''Esta función esta encargada de importar un modelo de Machine Learning que
    se encuentra como archivo de formato pickle (*.pkl) en un modelo que
    se pueda utilizar en python'''

    #cargar/importar el archivo con un nombre desde una carpeta determinada
    modelo = pickle.load(open(path + nombre_de_archivo, 'rb'))

    return modelo
```

Código 9: Exportación e importación de modelos de *Machine Learning* en archivos de formato pickle (*.pkl).

4.2.4.2. Implementación de etapa: Evaluación del modelo

Para la evaluación de los modelos y su rendimiento bajo diferentes grupos de características se requiere que el análisis sea flexible, por ejemplo, si se requiere analizar un dataset con solo 2 características por *frame*, entonces corresponde modificar el dataset original de las 14 caracteristicas y filtrarlo de modo que solo queden las 2 que se desea analizar. La implementación de esta función se realizó de la siguiente manera:

```
1
2  def filter_dataframe_train_dataset(dataframe_training_dataset, FRAME_LENGTH,
3                                     HOP_LENGTH, longitud_señal,
4                                     lista_caracteristicas_filtrar):
5      '''Función encargada de filtrar el dataframe del training dataset según la
6      lista de características que se le pase como entrada (en forma de array)'''
7
8      # Se filtra las características según la lista que se paso como entrada de
9      # la función y se guarda el resultado en un variable auxiliar de nombre
10     # auxil_matrix
11     auxil_row=[]
12     auxil_matrix=[]
13
14     for row in range(0,len(dataframe_training_dataset)):
15         auxil_row=[]
16         for columna in range(0,7):
17             for elemento in lista_caracteristicas_filtrar:
18                 auxil_row.append(dataframe_training_dataset[[elemento]].\
19                                  values[row][columna])
20
21         auxil_matrix.append(auxil_row)
22
23     num_ventanas_señal=int((((longitud_señal*tasa_muestreo)-FRAME_LENGTH)/
24                             HOP_LENGTH)+1)
25
26     # Se define los headers o encabezados para las caracteristicas del Dataframe
27     # Se tendrá el mismo header varias veces, según el número de ventanas/frames
28     header_caracteristicas=lista_caracteristicas_filtrar * num_ventanas_señal
29
30     # Se convierte la variable auxiliar en un dataframe y se le pasa los headers
31
32     dframe_train_dataset_filtrado = pd.DataFrame(auxil_matrix,
33                                                  columns=header_caracteristicas)
34
35     # La columna de 'clase' será la misma sin importar las caracteristicas
36     # filtradas, por lo que se le pasará sin mdoficación al final del dataframe
37     dframe_train_dataset_filtrado['clase']=dataframe_training_dataset['clase']
38
39     # La función retorna el dataframe de un training dataset con las
40     # características filtradas según una lista felxible que se pasa como
41     # entrada de la función.
42     return dframe_train_dataset_filtrado
```

Código 10: Filtrado del *Dataframe* del *Training Dataset* según una lista de características determinada.

Así, si se desea un *Training Dataset* con solo la característica de Centroide Espectral se le debe pasar como entrada de la función "filter_dataframe_train_dataset" una lista con el nombre de esta característica, este nombre debe ser el mismo que tiene la columna de la característica correspondiente en el *Dataframe* del *Training Dataset* original. La lista sería la siguiente:

```
1  lista_caracteristicas_filtrar=['0_Spectral centroid']
```

Y el *Dataframe* resultante tendría las siguientes dimensiones:

- Num. de Filas: 220 (220 muestras)
- Num. de Columnas: 8 ((1 caracteristica x 7 ventanas) + 1 Columna de clase)

Para seguir con la evaluación de los distintos modelos en sí, es importante entender que, tal como se vio en la sección 2.5.4, además del patrón de características también se debe escoger los hiperparámetros con los que se entrenarán los modelos a evaluar. Esta tarea de afinar los hiperparámetros es ardua y mecánica, porque consiste en la combinación de muchos valores.

Por ello, para la evaluación de los modelos, se utilizó una herramienta de la librería de Scikit Learn, esta permite la prueba de distintas combinaciones de hiperparámetros que se le pasen en una lista y entrega como resultado los hiperparámetros que dan el modelo más óptimo, evaluado según el método de validación cruzada. Esta herramienta es ideal para este trabajo de investigación, puesto que el método de validación cruzada es el mismo que se diseño en la sección 3.3.4.2. Cabria recordar que el número de iteraciones de la validación cruzada elegida en esa sección fue de 10. A continuación se muestra la implementación de una función que utiliza la herramienta mencionada de Scikit Learn para la evaluación de un determinado *Training Dataset*, la primera función es para el modelo SVM y la segunda para el modelo KNN:

```
from sklearn.model_selection import GridSearchCV

def evaluacion_training_dataset_SVM(dataframe_training_dataset, iteraciones):
    ''' Función encargada de mostrar la evaluación de la efectividad de
    distintos modelos entrenados con el algoritmo SVM, bajo distintas
    combinaciones de hiperparámetros, para un determinado Training
    Dataset. La evaluación se realiza con el método de validación cruzada'''

    # dataframe correspondiente a los vectores de caracteristicas (Es el
    # mismo dataframe de la entrada pero sin la columna "clase").
    dataframe_vectores_caracteristicas = dataframe_training_dataset.\
                                         loc[:,dataframe_training_dataset.\
                                             columns != 'clase']
    # dataframe correspondiente a la columna de clase (Corresponde a un
    # dataframe de una sola columna cuyo contenido se extrajo de la columna
    # "clase" del dataframe de entrada).
    dataframe_columna_clase=dataframe_training_dataset['clase']

    # Lista de hiperparamatros que el algoritmo GridSearchCV se encargara de
    # combinar
    parameters = {'C':[1, 5, 7, 8, 10, 15, 20, 35, 50, 80, 100, 120, 500,
                       700, 1000, 1200, 1500],
                  'gamma':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10,
                           0.00003, 0.0003, 0.003, 0.03, 0.3,
                           0.00005, 0.0005, 0.005, 0.05, 0.5,
                           0.00008, 0.0008, 0.008, 0.08, 0.8,
                          ]}

    # Prueba de todas las combinaciones posibles de la lista de
    # hiperparametros
    svc = svm.SVC()
    clf = GridSearchCV(svc, parameters, cv=iteraciones,
                       return_train_score=False)
    # Entrenamiento de distintos modelos bajo un training dataset definido
    clf.fit(dataframe_vectores_caracteristicas, dataframe_columna_clase)

    # Transformación de los resultados en un Dataframe para su fácil
    # visualización
    evaluacion_resultados=pd.DataFrame(clf.cv_results_)

    # Ordenar el datframe de las mejores combinaciones de hiperparametros
    # obtenidos
    evaluacion_resultados.sort_values(by='rank_test_score', inplace=True)
    evaluacion_resultados=[['param_C', 'param_gamma', 'mean_test_score',
                            'rank_test_score']]

    # El resultado es un dataframe que muestra el resultado de todas las
    # combinaciones posibles de hiperparametros ordenados por los que
    # tengan mejor rendimiento bajo evaluación de validacion cruzada de
    # un numero definido de iteraciones
    return evaluacion_resultados
```

Código 11: Evaluación de distintas combinaciones de hiperparámetros, para hallar el modelo más óptimo entrenado con el algoritmo SVM para un determinado *Training Dataset*.

```
def evaluacion_training_dataset_KNN(dataframe_training_dataset, iteraciones):
    ''' Función encargada de mostrar la evaluación de la efectividad de
    distintos modelos entrenados con el algoritmo KNN, bajo distintas
    combinaciones de hiperparámetros, para un determinado Training
    Dataset. La evaluación se realiza con el método de validación cruzada'''

    # dataframe correspondiente a los vectores de caracteristicas
    # (Es el mismo dataframe sin la columna "clase")
    dataframe_vectores_caracteristicas = dataframe_training_dataset.\
                                         loc[:,dataframe_training_dataset.\
                                             columns != 'clase']
    # dataframe correspondiente a la columna de clase
    # (Corresponde a un dataframe de solo la columna "clase")
    dataframe_columna_clase=dataframe_training_dataset['clase']

    # Lista de hiperparamatros que el algoritmo GridSearchCV se encargara de
    # combinar
    parameters = {'n_neighbors':[2,5,7,10,12,15,17,19,20,22,24,25,27,28,
                                 29,30,35,45,46,47,48,50],
                  'weights':['uniform', 'distance'],
                  'metric': ['euclidean', 'manhattan', 'minkowski']}

    # Prueba de todas las combinaciones posibles de la lista de
    # hiperparametros
    knn = neighbors.KNeighborsClassifier()
    clf = GridSearchCV(knn, parameters, cv=iteraciones,
                       return_train_score=False)
    # Entrenamiento de distintos modelos bajo un training dataset definido
    clf.fit(dataframe_vectores_caracteristicas, dataframe_columna_clase)

    # Transformación de los resultados en un Dataframe para su fácil
    # visualización
    evaluacion_resultados=pd.DataFrame(clf.cv_results_)

    # Ordenar el dataframe por las mejores combinaciones de hiperparametros
    # obtenidos (segun su valor de validacion cruzada).
    evaluacion_resultados.sort_values(by='rank_test_score', inplace=True)
    evaluacion_resultados=[['param_metric', 'param_n_neighbors', 'param_weights',
                            'mean_test_score', 'rank_test_score']]

    # El resultado es un dataframe que muestra el resultado de todas las
    # combinaciones posibles de hiperparametros ordenados por los que
    # tengan mejor rendimiento bajo evaluación de validacion cruzada de
    # un numero definido de iteraciones
    return evaluacion_resultados
```

Código 12: Evaluación de distintas combinaciones de hiperparámetros, para hallar el modelo más óptimo entrenado con el algoritmo KNN para un determinado *Training Dataset*.

En resumen, el código 10 implementado en esta sección, será de utilidad para poder hacer distintas pruebas, modificando las características del *Training Dataset* y de esta forma poder obtener un patrón de características que resulte en un modelo más óptimo. Los códigos 11 y 12, serán de utilidad en la evaluación automatizada de combinaciones de hiperparámetros, de modo que se pueda obtener la combinación que parámetros que resulten en el modelo más óptimo.

Un vez se conozca el patrón de características, los hiperparámetros y los algoritmos de *Machine Learning*, el entrenamiento de los modelos se debe realizar con las funciones implementadas en los códigos 7 (para SVM) y 8 (para KNN), para finalmente realizar la exportación de los modelos con el código 9. Recordar que la salida del Módulo de Entrenamiento del Modelo es un modelo/clasificador optimizado.

4.2.5. Implementación del módulo de Acondicionamiento de Señal para Procesamiento en Tiempo Real

4.2.5.1. Implementación de etapa: Sensado y Digitalización

Esta etapa se desarrolla de la misma forma que la sección 4.2.1.1, por tanto, ambas etapas comparten la misma implementación.

4.2.5.2. Implementación de etapa: Transmisión por la Red

El diseño de esta etapa y el de la sección 4.2.1.2 es igual hasta la parte donde se descarga la información, mientras en la sección 4.2.1.2 solo se requiere descargar los archivos mediante protocolo SFTP, para lo que respecta a este módulo, se requiere lectura de la señales de infrasonido en tiempo real, para ello se debe acceder al servidor *SeedLink* que se encuentra funcionando por defecto en el sensor. Este servidor funciona en el puerto 18000 del *Raspberry Pi*, por lo que para esta etapa se requeriría un cliente que se conecte y haga consultas continuas a dicho servidor. En ese sentido, la implementación de esta etapa se puede realizar por medio del cliente *SeedLink* que proporciona la librería *Obspy*, sin embargo, para este trabajo se decidió mejor utilizar la librería integrada *sl2influxdb*, esta librería realiza el trabajo de conexión y consulta al servidor *SeedLink*, y adicionalmente almacena la data consultada en una base de datos.

Como la parte del cliente de esta etapa y toda la siguiente etapa se encuentra ya implementada en forma conjunta con la librería *sl2influxdb*, se decidió detallar su funcionamiento en la siguiente sección.

4.2.5.3. Implementación de etapa: Almacenamiento de la Data en una Base de Datos en Tiempo Real

Antes de comenzar la implementación de esta etapa, se debe tener instalado y configurado el software de la base de datos con la que se trabajará. La base de datos que se decidió

utilizar en la sección de diseño es *influxDB*, por lo que se instaló esta base de datos en la computadora donde se almacenará la data. Se hizo el procedimiento de instalación, según la documentación, ejecutando las siguientes líneas en la interfaz de consola *Powershell* de *Windows*:

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-1.8.10_windows_amd64.zip
-UseBasicParsing -OutFile influxdb-1.8.10_windows_amd64.zip
Expand-Archive .\influxdb-1.8.10_windows_amd64.zip
-DestinationPath 'C:\Program Files\InfluxData\influxdb\'
```

Así, en la carpeta donde se instaló se puede observar los siguientes archivos:

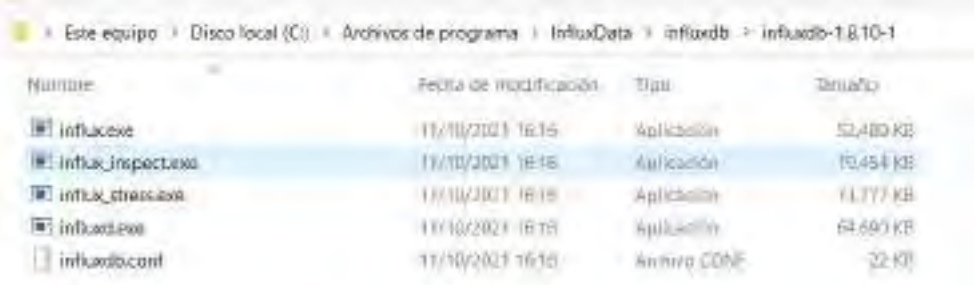

Figura 70: Archivos que forman parte de la instalación de *influxDB* en *Windows*.

Fuente: Propia

De los archivos que se ve en la figura 70, los que más nos interesan son “influxd.exe” y “influxdb.conf”, el primero es un ejecutable que hace que se ejecute *influxDB* en la computadora (servidor) y el segundo es el archivo de configuración. Para este trabajo no se realizó ninguna configuración, ya que la que se encuentra por defecto es suficiente. En la configuración por defecto, *influxDB* corre su servicio HTTP por defecto en el puerto 8086 de la computadora, es a través de este puerto que se puede realizar una conexión para hacer *querys* o para insertar registros en una base de datos.

Habiendo configurado la base de datos , la implementación de esta etapa se realizó utilizando la librería de *Python sl2influxdb*. Como se explicó en la anterior etapa, con esta librería se puede implementar de forma conjunta todo lo diseñado en la sección 3.3.5.3, en particular todo el esquema representado en la figura 54, desde la etapa de conexión al servidor *SeedLink* por medio de un cliente hasta la etapa de almacenamiento en la base de datos de *influxDB*.

Para la instalación de la librería, se deben descargar sus archivos del github del autor (https://github.com/marcopovitch/sl2influxdb), después se procede a instalar la librería desde la terminal de *Windows*, ubicándose en la carpeta descargada del github, y corriendo la instrucción “pip install .”, de esta forma, ya se puede utilizar la librería.

Según la documentación de la librería, para poder utilizarla se debe ejecutar una instrucción desde el terminal donde se le indica información sobre el servidor *SeedLink* y el servidor donde esta corriendo *influxDB*. La información que requiere *sl2influxdb* y que se le pasó en la instrucción es la siguiente:

- dbserver (nombre servidor donde esta corriendo *influxDB*): localhost
- dbport (port utilizado por servidor donde esta corriendo *influxDB*): 8086
- slserver (nombre servidor *SeedLink* en el *Raspberry Pi*): rs.local
- slport (port utilizado por servidor *SeedLink* en el *Raspberry Pi*): 180000
- streams (código único del sensor y código del *stream* de infrasonido): [(’AM’,’RA5FD’,’HDF’,’00’)]
- flushtime (cada cuanto tiempo en segundos se hace el envío de data a la base de datos): 3
- db (nombre de la base de datos *influxDB* donde se guardará la data, si no existe, se creará una nueva con este nombre): infra2
- keep (tiempo en días para la retención de data, 0 significa retención por tiempo indefinido): 0

Entonces, se hizo correr la instrucción con los campos llenados según la información indicada antes:

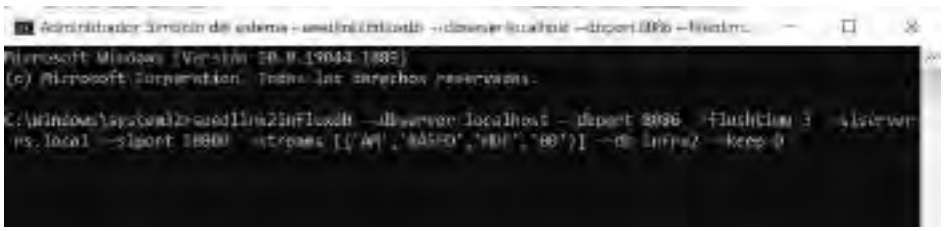

Figura 71: Librería *sl2influxdb* en funcionamiento.

Fuente: Propia

Según la documentación de la librería, no solo se almacena la data del sensor, sino también metadata de la latencia y retraso en el envío de paquetes *SeedLink*. Para este trabajo, solo nos interesa la data del infrasonido, la cuál se encuentra almacenada bajo los siguientes nombres de registros: "*value*" para el campo que alberga la amplitud de la señal de infrasonido, "*time*" para la marca de tiempo de cada punto de amplitud y "*counts*" para el nombre de la medición que alberga el campo anterior y su marca de tiempo .

Para comprobar que la data del sensor se esta copiando correctamente a la base de datos *influxDB*, se accede a tráves del CLI, para lo cual es necesario ubicarse en la carpeta de la instalación de *influxDB* a través del cmd de *Windows* y ejecutar el archivo "influx.exe". De esta forma se puede acceso al CLI de *influxDB* y se puede hacer consultas a la base de datos en un lenguaje muy parecido a SQL. Finalmente, se consultó los diez últimos registros de la base de datos, esta consulta se hizo indicándole el nombre de la base de datos y el nombre de la medición. En la figura 72 se muestra como se comprobó la existencia de los últimos 10 registros de la medición "*count*".

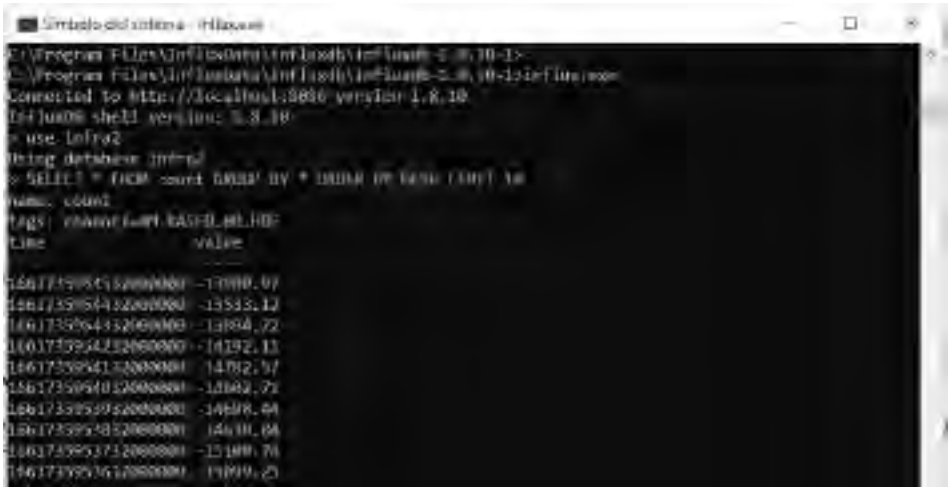

Figura 72: *Query* para consultar los últimos 10 registros de la medición "*count*" en la base de datos llamada "infra2", notar que la marca de tiempo se encuentra en formato de tiempo UNIX.

Fuente: Propia

4.2.5.4. Implementación de etapa: Extracción de Fragmentos de Análisis en Tiempo Real

La primera parte de esta etapa, tal como se explicó en la sección 3.3.5.4, requiere que se

hagan consultas continuas a la base de datos, esto se hizo por medio de un *script TICKscript* al que se nombro “udf_test.tick”. *TICKscript* es un lenguaje tipo SQL para realizar consultas a bases de datos *influxDB*. Esta consulta continua es del tipo *batching* y se implementó de la siguiente forma:

```
1  //conexión a la base de datos con nombre "infra2" y con política de retención
2  //de nombre "in_days"
3  dbrp "infra2"."in_days"
4
5  var data = stream
6      |from()
7          .measurement('count')
8      |window()
9          .period(3s)
10         .every(3s)
```

Código 13: Implementación del *batching* en el archivo *TICKscript*

Notar que aunque la consulta del código 13 es del tipo *batching*, en el *script* dice *stream*, esto no es un error, sino otra forma de implementar un *batching*, en este caso el *batching* se realizó en bloques de 3 segundos cada 3 segundos. Recordar que los fragmentos de análisis, según diseño, deben ser extraídos en bloques de 10.24, cada 2 segundos (8.24 segundos de solapamiento), por lo que en teoría los fragmentos debían extraerse con este criterio en este mismo *script TICKscript*, sin embargo, debido a los retrasos en la llegada de los paquetes de la señal se decidió dejar la implementación de la extracción de fragmentos y su solapamiento para la segunda parte de esta etapa (recordar de la sección 3.3.5.3 que cada paquete *SeedLink* se envía cada 2-3 segundos desde el servidor). Así, la única función de este *script* es la de hacer consultas o *querys* continuos de los registros más recientes de la base de datos *influxDB*.

Para la segunda parte de esta etapa se realizó la implementación del procedimiento de extracción de fragmentos de 10.24 segundos con solapamiento de 8.24 segundos (según diseño de la sección 3.2.4) y también el acondicionamiento de la señal para procesamiento en tiempo real, para ello se utilizó el software *Kapacitor*. Se hizo el procedimiento de su instalación, según la documentación, ejecutando las siguientes líneas en la interfaz de consola *Powershell* de *Windows*:

```
wget https://dl.influxdata.com/kapacitor/releases/kapacitor-1.6.5_windows_amd64.zip
-UseBasicParsing -OutFile kapacitor-1.6.5_windows_amd64.zip
Expand-Archive  .\kapacitor-1.6.5_windows_amd64.zip
-DestinationPath 'C:\Program Files\InfluxData\kapacitor\'
```

Así, en la carpeta donde se instaló se observa los siguientes archivos:

Figura 73: Archivos que forman parte de la instalación de *Kapacitor* en *Windows*.

Fuente: Propia

De los archivos que se ve en la figura 73, los que más nos interesan son "kapacitord.exe" y "kapacitor.exe", el primero es un ejecutable que hace que se ejecute *Kapacitor* en la computadora (servidor) y el segundo es un ejecutable que opera el CLI de *Kapacitor* para programar tareas o configuraciones a través de una terminal. Habiendo instalado todo lo necesario, se procedió con la configuración del software, para esto se ubica la carpeta donde están instalados los archivos de *Kapacitor* desde una terminal cmd de *Windows* y se ejecuta la siguiente instrucción para generar un archivo de configuración.

```
.\kapacitord.exe config > kapacitor.gen.conf
```

El archivo de configuración es muy importante para poder ejecutar exitosamente *Kapacitor*, mucha información importante se coloca aquí: el nombre del host donde se ejecuta *Kapacitor*, el puerto donde escucha el servidor API HTTP para escritura de *Kapacitor*, la información con respecto al servidor y el puerto donde esta corriendo *influxDB*, la ruta donde esta instalado *Python*, entre otros. En este archivo también se coloca cual será el *script* de *Python* que se utilizará en *Kapacitor*. La implementación de este *script* de *Python* se explicará aún más adelante, pero para propósitos de completar la instalación, de igual forma, se irá añadiendo información sobre este en el archivo de configuración, el *script Python* se llamará "udf_test.py" y se ubicará en la carpeta de

instalación de *Kapacitor*. Otra cosa a tomar en cuenta, es que para que *Kapacitor* pueda funcionar correctamente con archivos *Python* en *Windows* se le debe pasar un módulo que se encuentra en la página de github de *Kapacitor* (https://github.com/influxdata/kapacitor), para ello se descarga los archivos de la ruta "/udf/agent/py" del github en una carpeta de la computadora donde esta funcionando *Kapacitor* y se indica la ruta de esta carpeta en el archivo de configuración. El archivo de configuración tiene por nombre "kapacitor.gen.conf" y todo su contenido completado como se explicó en este párrafo se encuentra en el anexo 04.

La sección "udf" del archivo de configuración se refiere a "*User Defined Functions*" o "Funciones Definidas por el Usuario", que son funciones o *scripts* que se implementan por el usuario y que se hacen correr en *Kapacitor* para poder realizar procesamiento en tiempo real. Este *script* es al que se hacía referencia en el anterior párrafo y debe estar escrito en lenguaje *Python*. Algo importante a aclarar, es que, para que este *script* (también llamado "agente udf") funcione de forma integrada a *Kapacitor*, debe tener una estructura definida que se encuentra en la documentación del mismo software. Notar también que en el archivo de configuración del anexo 04 se indicó el nombre "deteccionAvalancha" en la sección udf, este es el nombre del agente udf que se debe implementar, este nombre es importante para enviar data desde la consulta continua que se implementó en *TICKscript* en el código 14. Este *script TICKscript* que realiza el *batching* se implementó en el código 13, pero no se hizo su conexión a *Kapacitor*, para poder enviar la data del *batching* a *Kapacitor* simplemente se le indica en el código el nombre y en que campo se almacenará esa data en *Kapacitor*. De esta forma, el archivo *TICKscript* modificado resulta de la siguiente forma:

```
dbrp "infra2"."in_days"

var data = stream
    |from()
        .measurement('count')
    |window()
        .period(3s)
        .every(3s)

data
    @deteccionAvalancha()
        .field('value')
```

Código 14: Modificación del archivo *TICKscript* para añadir la implementación del envío de data a *Kapacitor*

Habiendo implementado el *script* de *batching* en *TICKscript* que envía data al agente udf de nombre “deteccionAvalancha”, lo que sigue es la implementación de dicho udf. El udf se implementa basado en una estructura determinada según la documentación de *Kapacitor*, esta estructura se armará en un archivo *Python* con nombre “udf_test.py”, como ya se indicó dos párrafos más arriba. La estructura básicamente consiste en una clase de *Python* con constructor, atributos y métodos, su función es la de captar data a partir de una consulta continua (implementada en *TICKscript*) para posteriormente poder procesarla. Este agente tiene varios métodos, pero el que más nos interesa es el que se llama “*point*”, es en esta función donde se realiza el procesamiento de los puntos que van llegando de los *batchings* hacia *Kapacitor* y es donde se debe implementar todo el procesamiento de señales. La estructura del agente udf se puede observar en el diagrama de la figura 74 y el código completo en lenguaje *Python* se encuentra en el anexo 05.

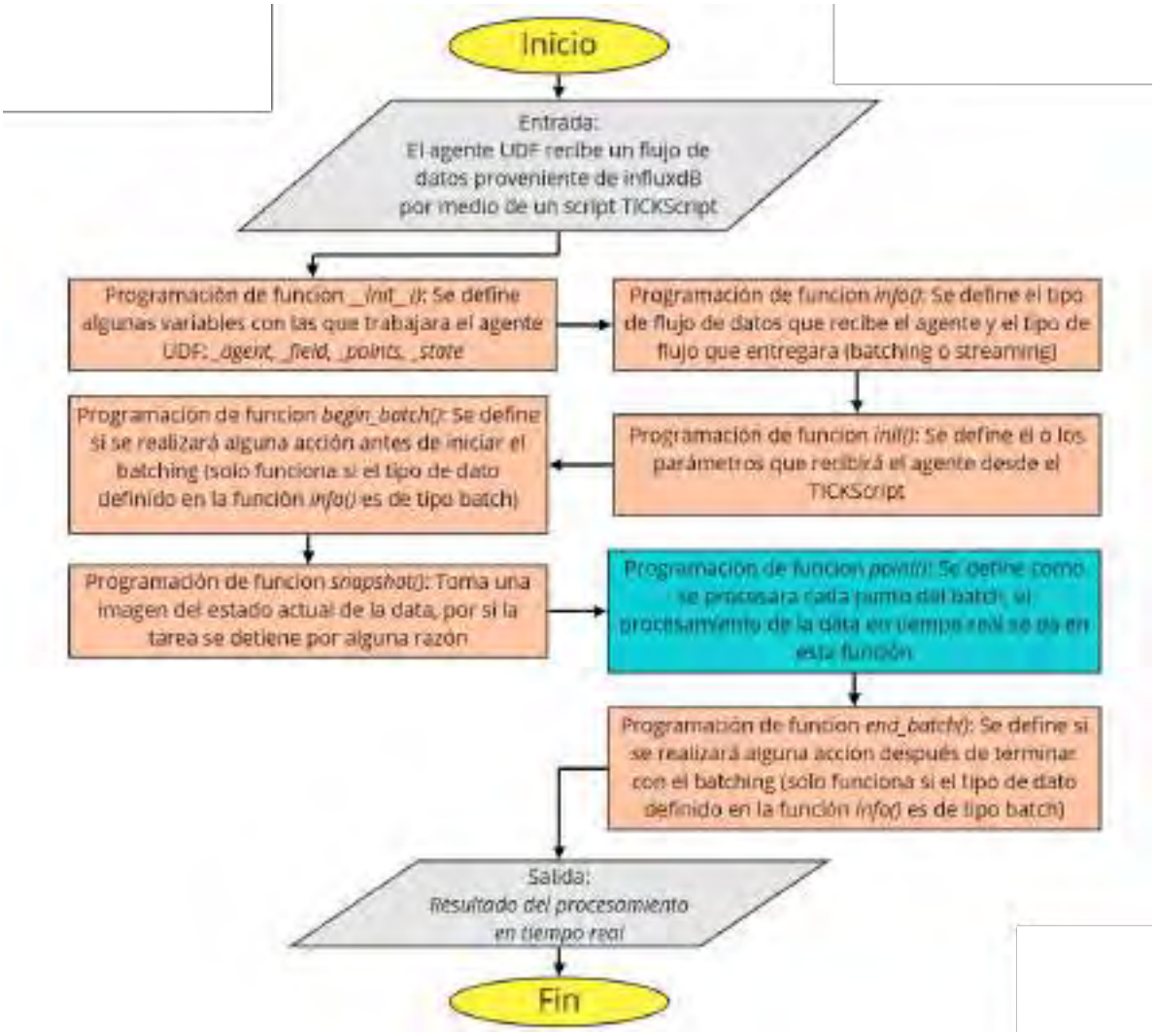

Figura 74: Estructura del agente udf según la documentación de *Kapacitor*.

Fuente: Propia

Como se puede observar en el agente udf, todo el procesamiento se realiza en la función "*point*", de esta forma, toda la implementación de los módulos de procesamiento del sistema se colocan dentro de esa función, también es allí donde se implementó el llenado de data de los fragmentos y el solapamiento que se diseñó en la sección 3.2.4. Al implementar los fragmentos en la función "*point*", se puede procesar la señal en bloques de 10.24 segundos con solapamiento de 8.24 segundos de forma precisa, incluso si los paquetes son enviados cada 3 segundos desde el servidor *SeedLink* en el *Raspberry Pi*.

Finalmente, aunque solo es propósito de este módulo el acondicionamiento de señal y no el procesamiento, es relevante mencionar los pasos para crear una "tarea" o "task" de

Kapacitor para que funcione de forma continua el procesamiento en tiempo real cuando ya se tenga implementado todo el sistema. Para ello, se debe ejecutar *Kapacitor* ubicándose en su carpeta de instalación desde un terminal cmd y haciendo correr el software *Kapacitor* indicándole las configuraciones realizadas en su archivo de formato "conf":

```
.\kapacitord.exe -config .\kapacitor.gen.conf
```

Luego, en otra terminal, se debe definir una tarea indicándole el nombre que se le asignará, además de indicarle también el nombre del archivo *TICKscript* respectivo, el cuál debe encontrarse en la carpeta de instalación de *Kapacitor*:

```
.\kapacitor.exe define tareaProcesamientoTiempoReal -tick .\udf_test.tick
```

Y por último, se activa la tarea:

```
.\kapacitor.exe enable tareaProcesamientoTiempoReal
```

4.2.6. Implementación del módulo de Clasificación de Eventos

El módulo de clasificación de eventos solamente requiere como entradas el vector de características del fragmento de análisis y el modelo optimizado que se obtuvo en la sección 4.2.4.2. Este modelo se encargará de clasificar los eventos para determinar si son avalanchas o no. La salida del módulo es un 1 si se determina que el fragmento analizado corresponde a un evento es avalancha y un 0 cuando no. Tener en cuenta que este módulo esta diseñado para que funcione en tiempo real con una llegada constante de fragmentos por lo que debe implementarse como parte del procesamiento en el agente udf de *Kapacitor* (ver figura 74). La función que realiza la clasificación se muestra en el código 15.

```
import pickle

def clasificacion_eventos(vector_caracteristicas, modelo):
    '''Realiza una clasificación a partir de un modelo y el vector de
    características de un fragmento de análisis en formato array'''

    # la cantidad de elementos del vector de características debe ser
    # la misma cantidad de los vectores de características con los que se
    # entreno el modelo (Número de características * número de
    # frames)

    array_caracteristicas = x.reshape(-1,len(vector_caracteristicas))

    # Etiqueta de clasificación, puede ser 1 o 0
    clasificacion = modelo.predict(array_caracteristicas)

    return clasificacion
```

Código 15: Clasificación de eventos según el modelo de *Machine Learning* que se le pase de entrada

El modelo que se le pasa como entrada a la función del código 15 debe haber sido cargado previamente con la función implementada en el código 9. Una vez se haya acoplado el código 15 al agente udf de *Kapacitor*, se obtiene el resultado de la clasificación de eventos en tiempo real a nivel de código, para poder almacenar este resultado en la base de datos se deben hacer algunas modificaciones al agente udf y también al archivo *TICKscript* (a partir de la modificación que ya se hizo en el código 14).

Las modificaciones al agente udf consisten en añadir un grupo de instrucciones que capturan la data a partir del flujo de datos que llega a *Kapacitor* (*batching*), la data más importante que se captura es la que indica la marca de tiempo. Cada punto llega a *Kapacitor* desde *influxDB*, y como ya se vio en la sección 4.2.5.3, viene con una estructura de clave-valor llamada “campo”, que almacena el valor de amplitud y la marca de tiempo por cada punto, el punto del que se capturó toda esta información es el que pertenece al último punto de cada fragmento de señal a analizar. El valor más importante a capturar es la marca de tiempo, la cual se encuentra bajo el nombre “*time*”. También se debe crear una medición nueva de nombre “detecciones” para albergar los campos del resultado de la clasificación que se genera en tiempo real mientras funciona *Kapacitor*. La implementación de la captura y envío de la data se observa en el código del anexo 05 (líneas 81 a 92). La data explicada anteriormente se envía una vez por cada fragmento a *influxDB* por medio del *TICKscript*, por ello, también se debe modificar el *TICKscript* para captar la información de la clasificación y su marca de tiempo proveniente de *Kapacitor*, entonces se añade el

nombre “detecciones” como un argumento en el archivo *TICKscript* para que el almacenamiento en la base de datos *influxDB* pueda hacerse efectiva. La modificación final del archivo *TICKscript* se puede observar en el código 16.

```
dbrp "infra2"."in_days"

var data = stream
    |from()
        .measurement('count')
    |window()
        .period(3s)
        .every(3s)

data
    @deteccionAvalancha()
        .field('value')
    |InfluxDBOut()
        .database('infra2')
        .measurement('detecciones')
        .retentionPolicy('in_days')
```

Código 16: Modificación final del archivo *TICKscript*, este *script* envía un flujo de datos de tipo *batching* a *Kapacitor* y también recibe data en un flujo constante desde *Kapacitor*

Para comprobar que la data de la clasificación se esta almacenando de forma correcta se realiza un *query* a la base de datos con el nombre de la medición con la que se le envía desde el udf de *Kapacitor* (“detecciones”). La comprobación se observa en la figura 75.

Figura 75: *Query* para consultar los últimos 10 registros de la medición “detecciones” en la base de datos llamada “infra2”, notar que la marca de tiempo se encuentra en formato de tiempo UNIX.

Fuente: Propia

4.2.7. Implementación del módulo de Visualización del Resultado

Según lo diseñado, se utilizó Grafana para poder visualizar el resultado de clasificación mediante una gráfica en tiempo real. Para su implementación lo primero que se hizo fue instalar el software. En la figura 76 se puede observar algunos archivos de la carpeta de instalación.

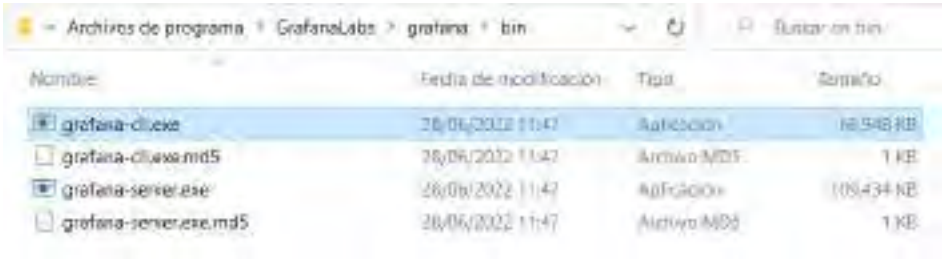

Figura 76: Archivos que forman parte de la instalación de Grafana en *Windows*.

Fuente: Propia

El archivo “grafana-server.exe” de la figura 76 es muy importante, ya que al ejecutarlo se hace correr el programa de Grafana en el servidor local corriendo en el port HTTP 3000. Sin embargo, antes de iniciar el programa es necesario configurar una características de las gráficas que no se puede realizar en la interfaz gráfica. En el archivo “custom.ini”, cuya ubicación se ve en la figura 77, se debe descomentar y modificar la variable de “min_refresh_interval” de modo que su nuevo valor sea 500ms, esto se hace así, ya que, si bien las gráficas de Grafana tienen el tiempo de actualización de la gráficas modificables, el tiempo mínimo es 5 segundos, para este trabajo se requiere una actualización lo más cercana posible al tiempo real, en ese sentido, se configura que el tiempo mínimo de actualización de gráficas dentro de las opciones sea 500 ms.

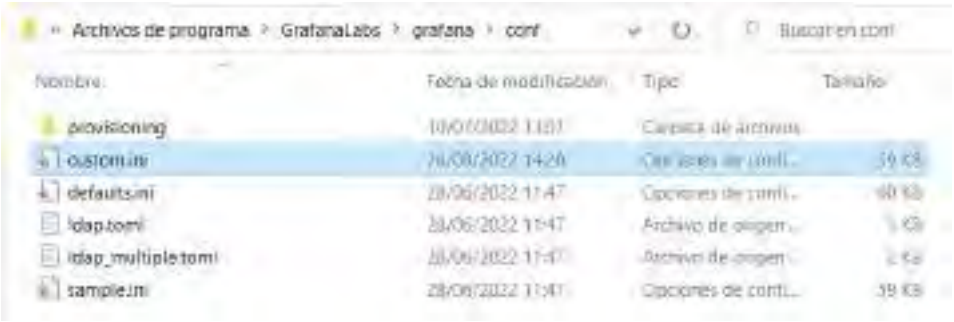

Figura 77: Archivos de configuración para Grafana, las modificaciones a la configuración se hacen en “custom.ini”.

Fuente: Propia

Lo demás de la configuración se puede hacer desde la interfaz gráfica de Grafana, para lo cual se debe ejecutar el archivo “grafana-server.exe” que se ve en la figura 76. Al hacer correr el ejecutable podemos acceder a la interfaz gráfica desde la URL “localhost:3000”, en la figura 78 se observa la pantalla de inicio de la interfaz gráfica.

Figura 78: Pantalla de inicio de la interfaz gráfica de Grafana.

Fuente: Propia

Luego, se procedió a configurar la conexión a la base de datos *influxDB*, esto se hizo desde la URL “localhost:3000/datasources/edit/rmfrSQWVk”, donde lo único que se llena es la URL por donde se accede al servidor de la base de datos, que en este caso es

“localhost:8086”, y el nombre de la base de datos que, para este trabajo, es “infra2”.

El siguiente paso es crear el *Dashboard* donde se debe graficar la data de la base de datos de *influxDB*, esto se hace desde la URL “localhost:3000/dashboard/new?orgId=1”. Luego de crear el *Dashboard* se prosiguió a configurar las gráficas. La primera gráfica que se configuró corresponde al de la señal de infrasonido, esta configuración se puede observar en la figura 79. La segunda gráfica que se configuró corresponde al resultado de la clasificación, que esta compuesto de solo ceros y unos, esta configuración se puede observar en la figura 80.

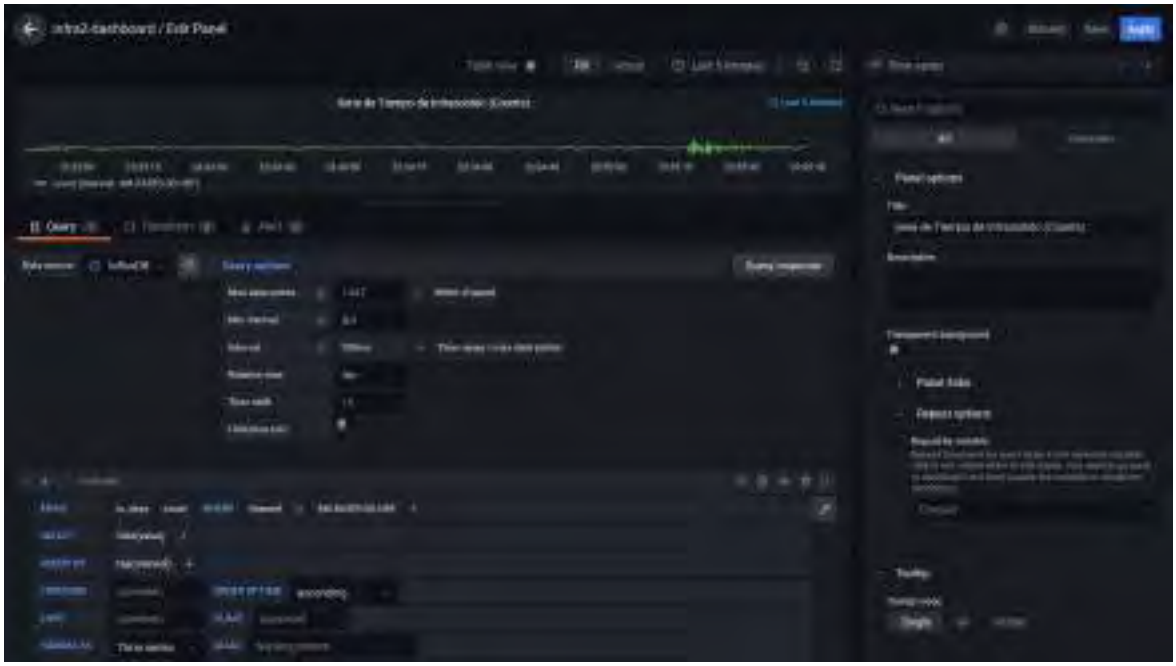

Figura 79: Configuración para la gráfica de la señal de infrasonido.

Fuente: Propia

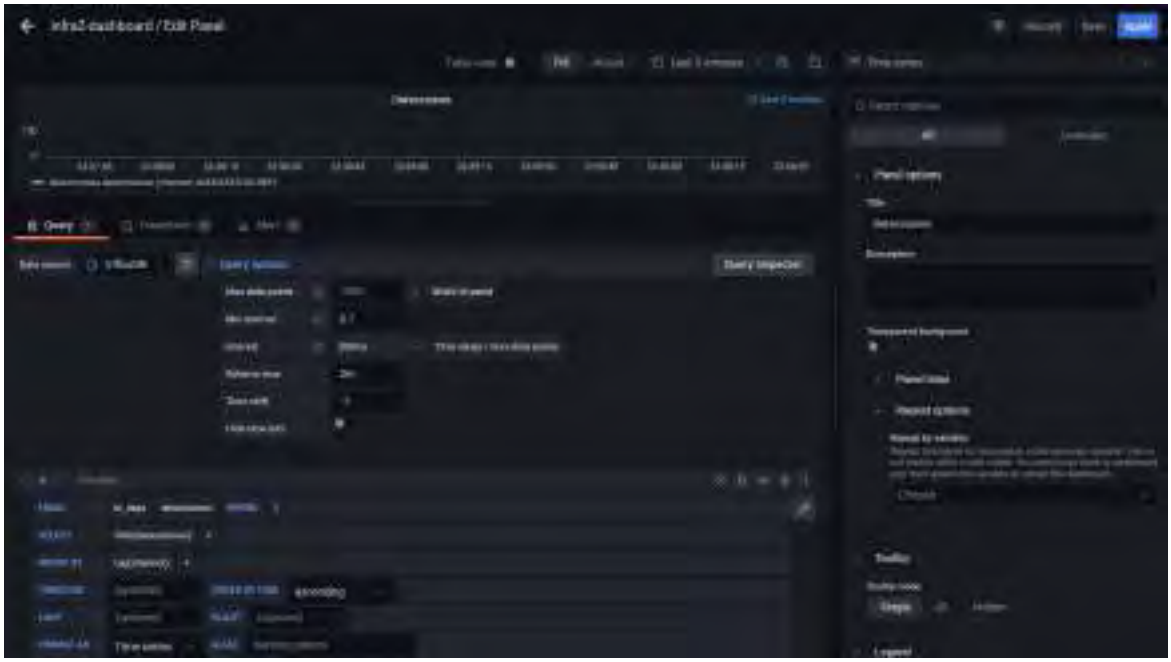

Figura 80: Configuración para la gráfica del resultado de clasificación.

Fuente: Propia

El paso final es ordenar las gráficas en el *Dashboard* resultante, Grafana ofrece una fácil organización de simple arrastre y modificación de tamaño de las gráficas de forma muy intuitiva. De esta forma, el resultado final del *Dashboard* se observa en la figura 81.

Figura 81: Dashboard de Grafana donde se pueden observar las gráficas configuradas: Los registros de detección y la señal de infrasonido, ambos se pueden observar en tiempo real.

Fuente: Propia

Una cosa a tomar en cuenta, es que Grafana hace consultas a la base de datos cada cierto intervalo de tiempo configurable para poder actualizar las gráficas del *Dashboard*, este tiempo se puede configurar desde el mismo *Dashboard* pero por defecto solo se puede configurar desde 5 segundos a más, es por esto que se configuro el tiempo mínimo como 500 ms en el archivo "custom.ini" en la primera parte de este módulo. El intervalo de tiempo de actualización se puede observar en la parte superior derecha de la figura 81, para este trabajo se configuro para que todas las gráficas se actualicen cada 1 segundo.

4.3. Diagramas de interacción de códigos

Finalmente, corresponde indicar como interactúan todos los códigos y herramientas implementados en la sección 4.2.

En primer lugar, podemos observar en la figura 82 el diagrama de interacción para la obtención del modelo más óptimo, que es la primera parte del sistema de detección del presente trabajo.

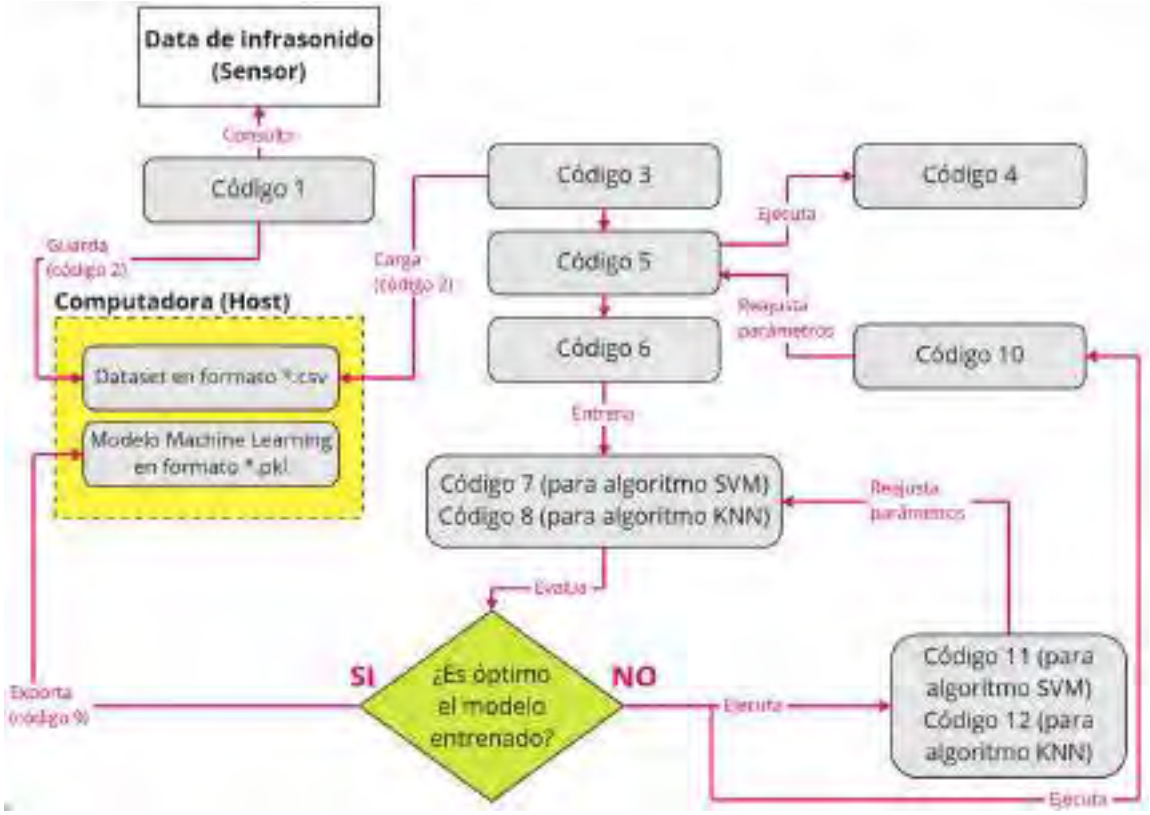

Figura 82: Diagrama de interacción de códigos para la obtención del modelo más óptimo.

Fuente: Propia

En segundo lugar, podemos observar en la figura 83 el diagrama de interacción para la detección de eventos en tiempo real, esta etapa corresponde a la segunda parte del sistema de detección del presente trabajo. Cabe la aclaración que las "mediciones" almacenadas en la base de datos que aparecen en el diagrama, corresponden a la forma en como se almacenan los registros en *influxDB* (ver sección 3.3.5.3), la medición 1 corresponde a la data de infrasonido del sensor y tiene por nombre "*count*" mientras que la medición 2 corresponde a la data del resultado de la detección en tiempo real y tiene por nombre "detecciones".

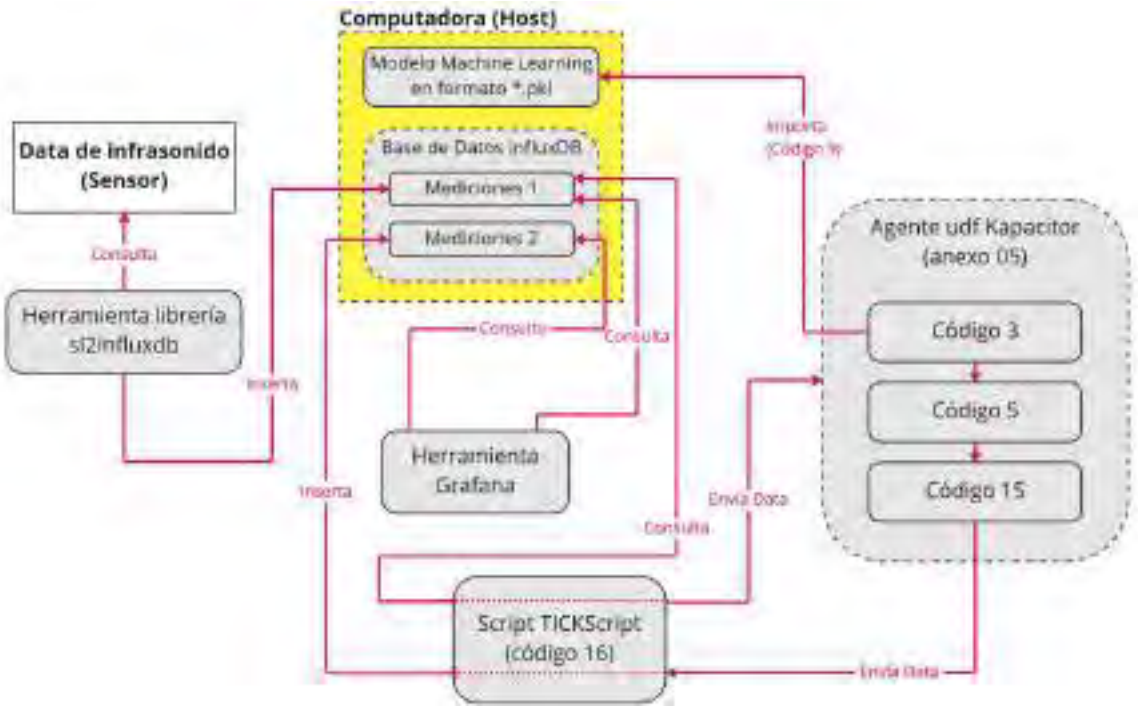

Figura 83: Diagrama de interacción de códigos para la detección de eventos en tiempo real.

Fuente: Propia

V. Pruebas y Resultados

5.1. Pruebas y Resultados del módulo de Extracción de Características

El resultado de la implementación del módulo en la sección 4.2.3, corresponde a un vector compuesto del patrón de características. Para poder encontrar el patrón de características se requiere realizar algunas pruebas con las características de las muestras del dataset obtenido en el módulo de recolección de datos. Para ello, se hizo el procedimiento de *scattering*, de modo que se pueda encontrar las características que tengan demasiada correlación.

En la figura 84 se muestra el análisis de *scattering* para algunas características que se observó que aportan la misma información, este procedimiento se realizó para todos las características, comparando de forma visual una frente a otra otra. El resultado del análisis para todas las características se puede observar en la tabla 5.

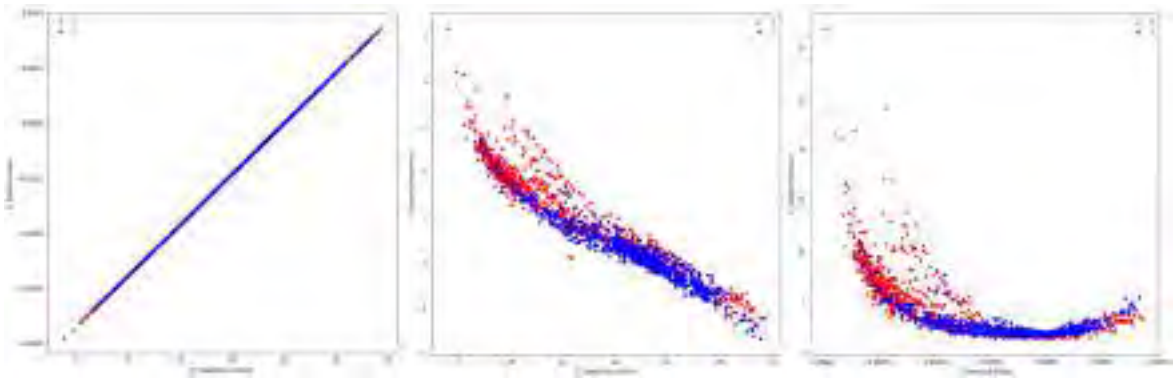

Figura 84: Análisis de *scattering* para tres pares de características.

Fuente: Propia

En la tabla 5 se muestran los resultados del análisis con *scattering* de las 14 características. Se marcó con una X de color rojo los pares de características que se observó tienen un alto nivel de correlación.

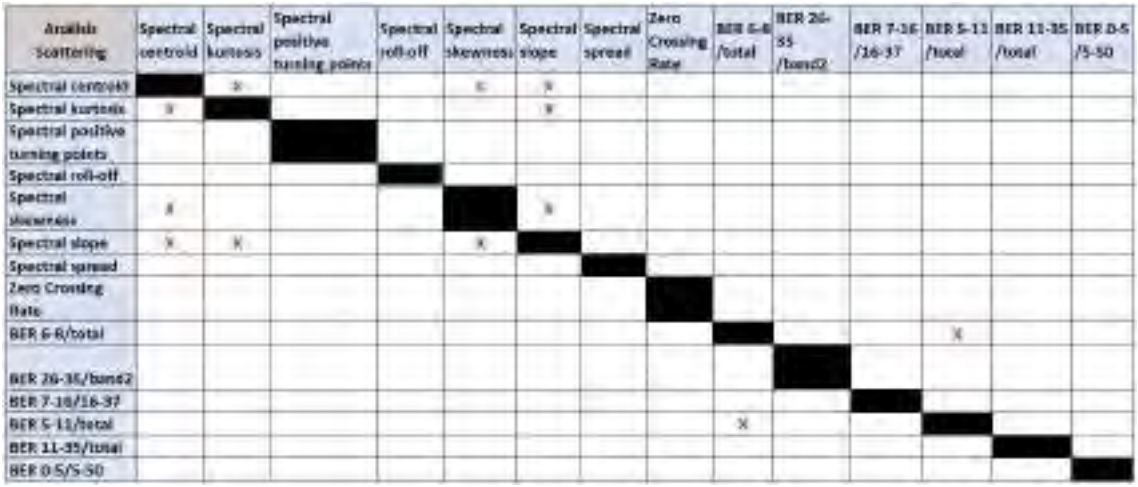

Tabla 5: Resultado del análisis de *scattering* para las 14 características iniciales.

Fuente: Propia

Del análisis se observa que el centroide espectral y la pendiente espectral son las dos características que menos información aportan respecto a las demás, dado que ambos tienen un alto nivel de correlación con hasta 3 características.

5.2. Pruebas y Resultados del módulo de Entrenamiento del Modelo

El resultado de la implementación de este módulo módulo corresponde a un modelo de *Machine Learning* optimizado (ver sección 4.2.4). Para ello, se procedió con la evaluación de distintas combinaciones de características bajo los modelos de SVM y KNN. En la figura 85, se muestra un ejemplo del resultado de evaluar la combinación de hiperparámetros para dos modelos entrenados con el *Training Dataset* original de las 14 características, la primera tabla de la figura representa el resultado de emplear el código 11 y la segunda, el resultado de emplear el código 12.

Evaluación SVM
(10 mejores combinaciones de hiperparametros) :

	param_C	param_gamma	mean_test_score	rank_test_score
315	1000	0.00003	0.840908	1
253	700	0.00003	0.840908	1
95	10	0.00003	0.840908	1
337	1200	0.00003	0.840908	1
359	1500	0.00003	0.840908	1
271	800	0.00003	0.840908	1
176	50	0.00001	0.840908	1
294	500	0.00001	0.830304	8
193	50	0.00008	0.831818	9
189	50	0.00003	0.831818	9

Evaluación KNN
(10 mejores combinaciones de hiperámetros):

	param_metric	param_n_neighbors	param_weights	mean_test_score	rank_test_score
44	manhattan	2	uniform	0.754545	1
69	manhattan	27	distance	0.740909	2
67	manhattan	25	distance	0.736364	3
71	manhattan	28	distance	0.736364	4
131	minkowski	50	distance	0.731818	5
42	euclidean	50	uniform	0.731818	5
43	euclidean	50	distance	0.731818	5
11	euclidean	15	distance	0.731818	5
130	minkowski	50	uniform	0.731818	5
35	euclidean	45	distance	0.731818	5

Figura 85: Resultados de las 10 mejores combinaciones de hiperparametros para modelos SVM y KNN, el *Training Dataset* utilizado en este ejemplo es el que se compone de las 14 características originales.

Fuente: Propia

Como se observa en la figura 85, el mejor modelo de SVM que se puede obtener utilizando el *Training Dataset* con las 14 características tiene una efectividad (*accuracy*) de 84.09 % (0.8409), mientras que la efectividad (*accuracy*) del mejor modelo de KNN en estas circunstancias es de 75.45 % (0.7545). Este valor del porcentaje de efectividad (*accuracy*) será el que se usará como criterio de evaluación para la búsqueda del patrón de características y para la elección del modelo más óptimo.

Luego, se realizaron más pruebas, pero esta vez utilizando para el entrenamiento unos *Training Datasets* compuestos de características individuales aisladas. En la tabla 6 se observa el análisis con las efectividad de clasificación de los mejores modelos SVM y KNN que se pudieron obtener para distintos *Training Dataset* compuestos de cada característica individual.

Característica	*Eficiencia Mejor Modelo SVM (%)*	*Eficiencia Mejor Modelo KNN (%)*
Spectral centroid	75.91	72.27
Spectral kurtosis	82.27	75.45
Spectral positive turning points	74.54	69.55
Spectral roll-off	81.82	73.64
Spectral skewness	82.27	78.64
Spectral slope	72.27	72.27
Spectral spread	82.27	75.45
Zero Crossing Rate	76.82	75
BER 6-8/total	63.18	64.55
BER 26-35/total	75.91	75.45
BER 7-16/16-37	82.73	78.64
BER 5-11/total	67.27	66.82
BER 11-35/total	79.55	77.27
BER 0-5/5-50	68.63	70.91

Tabla 6: Efectividad de los mejores modelos para características individuales. Se ven resaltados en amarillo las características con mejores resultados.

Fuente: Propia

Bajo el análisis de la tabla 6 las características más relevantes son: Kurtosis Espectral, Roll-off Espectral, Asimetría Espectral, Envergadura Expectral y la Relación de energía de la banda de 7-16 Hz en relación a la banda de 16-37 Hz. Estas superan el umbral del 80 % de efectividad para el algoritmo SVM y del 75 % para KNN.

Después de haber encontrado el primer grupo de las 5 características más relevantes, se hizo un segundo análisis tomando en cuenta las pruebas realizadas en la sección anterior. Además, según se observa en la tabla del análisis de *scattering* (ver tabla 5), ninguna de las 5 características tiene correlación entre si, por lo que todas aportan información distinta y pueden formar un primer grupo preliminar del patrón de características. En esa misma tabla se observa que la características de *Spectral Centroid*, *Spectral Slope* y *Spectral Skewness* tienen una alta correlación entre si, por tanto, al elegir la característica de *S. Skewness* como parte del patrón preliminar, las otras dos quedan descartadas. También se descartará la característica de BER 6-8/total, debido a que en el análisis de *scattering* se observó que tenía un alto nivel de correlación con la característica de BER 5-11/total, y ya que ambas aportan la misma información, se decidió quedarse con la característica que tenga más efectividad de las dos.

La siguiente prueba que se realizó corresponde a una comparación de la efectividad de

los modelos entrenados con un *Training Dataset* compuesto de la combinación del patrón preliminar más una de las otras características que restan. Este análisis se observa en la tabla 7 .

Combinación del patron de 5 mejores Características con:	*Eficiencia Mejor Modelo SVM (%)*	*Eficiencia Mejor Modelo KNN (%)*
Spectral positive turning points	91.36	80.45
Zero Crossing Rate	85.91	74.09
BER 26-35/total	87.73	79.55
BER 5-11/total	89.55	75.91
BER 11-35/total	90.45	76.82
BER 0-5/5-50	89.09	78.64

Tabla 7: Combinación de las 5 mejores características con las otras restantes. Se ven resaltados en amarillo las combinaciones con mejores resultados.

Fuente: Propia

De la tabla 7, se deduce que la mejor combinación de características es la que esta compuesta del patrón preliminar con la característica de Puntos de Giro Positivos Espectrales, existen también otras combinaciones características que superan el umbral del 90% de efectividad y otras que se acercan mucho al umbral, todas estas características se resaltaron en amarillo en la tabla. Teniendo en cuenta lo anterior, y considerando que las características del patrón preliminar y los resaltados en amarillo no presentan correlación, se decidió hacer una evaluación de efectividad entrenando un modelo con un *Training Dataset* compuesto de todas estas características. Las 9 características a probar son: Kurtosis Espectral, Puntos de Giro Positivos Espectrales, Roll-off Espectral, Asimetría Espectral, Envergadura Espectral, BER 7-16/16-37, BER 5-11/total, BER 11-35/total y BER 0-5/5-50.

La evaluación de las distintas combinaciones de hiperparámetros, con los que se obtienen los 10 mejores modelos, entrenados con el *Training Dataset* de las 9 características, se puede observar en la figura 86.

Figura 86: Resultados de las 10 mejores combinaciones de hiperparametros para modelos SVM y KNN, el *Training Dataset* utilizado para ambos modelos es el que se compone de las 9 características.

Fuente: Propia

De la figura 86 se deduce que, a diferencia del entrenamiento con las 14 características iniciales, el uso de las 9 características elegidas ha resultado en una mejora en la efectividad de aproximadamente un 6 % para ambos algoritmos.

A partir de aquí, se intentó hacer pruebas buscando más combinaciones de características con los cuáles se obtuviera un modelo con mejor efectividad, sin embargo, la efectividad obtenida con las 9 características era inmejorable, por ello, se decidió que estas serían las que conformen el patrón de características.

Recordar también que el resultado de este módulo debe ser un modelo/clasificador optimizado, por ello se decidió elegir un modelo por cada algoritmo, y habiendo definido ya el patrón de características, solo quedaría por definir cuáles serán los hiperparámetros elegidos.

En lo que respecta a los hiperparámetros del algoritmo SVM, si bien se hubiera podido elegir solo el par de hiperparámetros que generan el mejor modelo, hay que tener en cuenta que su parámetro *C* es bastante grande y su parámetro *gamma* es bastante pequeño, como se explicó en la sección 3.3.4.1 y como se observo en la figura 25, un valor medio en ambos parámetros es lo ideal, por lo que se decidió entrenar dos modelos, uno con el par de

parámetros del mejor modelo ($C = 500$ y $gamma = 0.00008$) y otro con los parámetros del segundo mejor modelo ($C = 15$ y $gamma = 0.0008$).

En lo que respecta a los hiperámetros del algoritmo KNN, se eligió solamente el mejor modelo cuya combinación de hiperparámetros son: $metric = euclidean$, $n\_neighbors = 2$ y $weights = uniform$.

En conclusión, las pruebas realizadas en este módulo nos deja como resultado lo siguiente:

- Composición del patrón de Características: *Spectral kurtosis*, *Spectral positive turning points*, *Spectral roll-off*, *Spectral skewness*, *Spectral spread*, BER 7-16/16-37, BER 5-11/total, BER 11-35/total y BER 0-5/5-50.
- Se entrenaron tres modelos: dos con el algoritmo SVM y uno con KNN, esto a fin de ser evaluados y comparados en la sección de pruebas y resultados del módulo de clasificación de eventos en tiempo real.
- Los hiperparámetros del primer modelo SVM son: $C = 500$ y $gamma = 0.00008$. Su efectividad de clasificación (*accuracy*) teórica es del 92.73 %.
- Los hiperparámetros del segundo modelo SVM son: $C = 15$ y $gamma = 0.0008$. Su efectividad de clasificación (*accuracy*) teórica es del 92.27 %.
- Los hiperparámetros del modelo KNN son: $metric = euclidean$, $n\_neighbors = 2$ y $weights = uniform$. Su efectividad de clasificación (*accuracy*) teórica es del 82.27 %.

5.3. Pruebas y Resultados del módulo de Clasificación de Eventos en Tiempo Real

Finalmente, se realizaron pruebas piloto con el sistema en tiempo real implementado en la laguna Palcacocha, para ello, se utilizaron 3 modelos para realizar las pruebas, los mismos tres modelos que resultaron de las pruebas y análisis realizados en la sección anterior. Para propósitos prácticos, se le asignó un nombre a cada modelo, de modo que quedaron de la siguiente forma:

- **Modelo SVM 1** : El modelo entrenado bajo el algoritmo SVM con los hiperparámetros: $C = 500$ y $gamma = 0.00008$.
- **Modelo SVM 2** : El modelo entrenado bajo el algoritmo SVM con los hiperparámetros: $C = 15$ y $gamma = 0.0008$.
- **Modelo KNN** : El modelo entrenado bajo el algoritmo KNN con los hiperparámetros: $metric = euclidean$, $n\_neighbors = 2$ y $weights = uniform$.

Antes de iniciar la evaluación de las pruebas, habría que recordar que el sistema de detecciones da como resultado una etiqueta de clasificación (1 para avalanchas y 0 para no-avalanchas) cada 2 segundos, sin embargo, la evaluación no se realizará etiqueta por etiqueta sino por evento. En este contexto, un evento básicamente es una clasificación o un conjunto de clasificaciones adyacentes (ya sea avalancha o ruido) que no corresponden a un periodo de silencio (periodo donde no existió ni avalanchas ni ruidos ni viento), esto se hizo ya que, si por ejemplo, el modelo predijo avalanchas en el segundo 2 y en el segundo 4 de cierta hora, esto no quiere decir que se detecto dos avalanchas, sino que son dos detecciones que pertenecen a un solo evento. Básicamente un evento inicia cuando termina un periodo de silencio y termina cuando inicia otro periodo de silencio.

En este sentido, se hicieron las pruebas cubriendo parte de dos días, desde las 7:00 P.M. del día 12 de mayo del 2022 hasta las horas 10:30 A.M. del día 13 de mayo del 2022. En este intervalo el sistema pudo detectar 133 eventos. Las pruebas se hicieron de manera presencial en la laguna Palcacocha, para así poder detectar de forma objetiva el número de avalanchas que sucedieron en ese intervalo. En la tabla 8 se observa un ejemplo de como se realizó el análisis del resultado de las pruebas (solo con el modelo SVM 2), en esta tabla se puede observar el análisis de 4 eventos, notar que el evento 11 se considera como una avalancha aunque el modelo haya predicho una avalancha durante solamente 2 segundos, siendo la etiqueta real del evento la de una no-avalancha, este evento se considera un falso positivo.

N° Evento	Hora de clasificación (UTC-5)	Etiqueta predicha (Modelo SVM 2)	Etiqueta predicha del evento (Modelo SVM 2)	Etiqueta real del evento	Falsos positivos	Falsos negativos	Verdadero Positivo	Verdadero Negativo
11	2022-05-13T14:50:35	0	1	0	X			
	2022-05-13T14:50:33	0						
	2022-05-13T14:50:31	0						
	2022-05-13T14:50:29	0						
	2022-05-13T14:50:27	1						
	2022-05-13T14:50:25	0						
	2022-05-13T14:50:23	0						
	2022-05-13T14:50:21	0						
	2022-05-13T14:50:19	0						
	2022-05-13T14:50:17	0						
	2022-05-13T14:50:15	0						
	2022-05-13T14:50:13	0						
	2022-05-13T14:50:11	0						
	2022-05-13T14:50:09	0						
	2022-05-13T14:50:07	0						
	2022-05-13T14:50:05	0						
12	2022-05-13T14:49:47	0	0	0				X
	2022-05-13T14:49:45	0						
	2022-05-13T14:49:43	0						
	2022-05-13T14:49:41	0						
	2022-05-13T14:49:39	0						
	2022-05-13T14:49:37	0						
	2022-05-13T14:49:35	0						
	2022-05-13T14:49:33	0						
	2022-05-13T14:49:31	0						
	2022-05-13T14:49:29	0						
40	2022-05-13T13:49:25	1	1	1			X	
	2022-05-13T13:49:23	1						
	2022-05-13T13:49:21	1						
	2022-05-13T13:49:19	1						
	2022-05-13T13:49:17	0						
89	2022-05-13T01:22:25	0	0	1		X		
	2022-05-13T01:22:23	0						
	2022-05-13T01:22:21	0						
	2022-05-13T01:22:19	0						
	2022-05-13T01:22:17	0						

Tabla 8: Ejemplo de análisis para un grupo de eventos clasificados por el sistema de detección (modelo SVM 2).

Fuente: Propia

Así como se hizo el análisis en la tabla 8, se procedió a realizarlo para todos los eventos y los tres modelos, resultando en las matrices de confusión que se observan en la siguiente tabla:

	Matriz de confusión					
	Falsos positivos	Falsos negativos	Verdadero Positivo	Verdadero Negativo	Efectividad Modelo	Total Eventos
Modelo SVM 1	52	1	24	56	0.60150376	133
Modelo SVM 2	43	1	24	65	0.66917293	133
Modelo KNN	48	0	25	60	0.63909774	133

Tabla 9: Matrices de confusión para los modelos evaluados.

Fuente: Propia

Es de notar el bajo nivel de efectividad en la tabla 9, ninguno de los modelos logra superar el 70 % de efectividad, esto se debe al criterio que se utilizó para considerar un evento como avalancha, tal como se vio en el evento 11 de esa tabla, se considero como avalancha al evento predicho por el modelo solo por una clasificación errónea. Para intentar mejorar este panorama, se decidió hacer pruebas utilizando un criterio distinto, el nuevo criterio consiste en que un evento predicho por el modelo se debe considerar avalancha siempre y cuando tenga dos o más clasificaciones de avalancha consecutivas, así, por ejemplo el evento 11 no sería más un evento de avalancha y en su métrica de efectividad dejaría de ser un error (falso positivo) para convertirse en un acierto (verdadero negativo).

En la tabla 10 se observa el análisis con el nuevo criterio para un par de eventos clasificados con el modelo SVM 2, en este ejemplo se gráfica claramente los pros y los contras del nuevo criterio, aunque el evento 11 ahora pasa a ser un acierto, pueden existir eventos como el 81 que aunque con el criterio anterior eran aciertos, ahora son errores. Sin embargo, aún incluso con las desventajas del nuevo criterio, en una evaluación más extensa se puede observar que la efectividad del sistema de detección se incrementa considerablemente, esto lo podemos observar en la matriz de confusión de la tabla 11.

N° Evento	Hora de clasificación (UTC+5)	Etiqueta predicha (Modelo SVM 2)	Etiqueta predicha del evento (Modelo SVM 2)	Etiqueta real del evento	Falsos positivos	Falsos negativos	Verdadero Positivo	Verdadero Negativo
11	2022-05-13T14:50:35	0	0	0				X
	2022-05-13T14:50:33	0						
	2022-05-13T14:50:31	0						
	2022-05-13T14:50:29	0						
	2022-05-13T14:50:27	1						
	2022-05-13T14:50:25	0						
	2022-05-13T14:50:23	0						
	2022-05-13T14:50:21	0						
	2022-05-13T14:50:19	0						
	2022-05-13T14:50:17	0						
	2022-05-13T14:50:15	0						
	2022-05-13T14:50:13	0						
	2022-05-13T14:50:11	0						
	2022-05-13T14:50:09	0						
	2022-05-13T14:50:07	0						
	2022-05-13T14:50:05	0						
81	2022-05-13T04:51:15	1	0	1		X		
	2022-05-13T04:51:13	0						
	2022-05-13T04:51:11	0						
	2022-05-13T04:51:09	0						
	2022-05-13T04:51:07	0						

Tabla 10: Ejemplo de análisis con el nuevo criterio para un par de eventos clasificados por el sistema de detección (modelo SVM 2).

Fuente: Propia

	Matriz de confusión					
	Falsos positivos	Falsos negativos	Verdadero Positivo	Verdadero Negativo	Efectividad Modelo	Total Eventos
Modelo SVM 1	22	3	22	86	0.81203008	133
Modelo SVM 2	16	2	23	92	0.86466165	133
Modelo KNN	28	2	23	80	0.77443609	133

Tabla 11: Matrices de confusión para los modelos evaluados utilizando el nuevo criterio de detección.

Fuente: Propia

De la tabla 11, se deduce, que el nuevo criterio de detección es mucho más óptimo. Con respecto a los algoritmos, también se observa que aunque ambos tienen similares resultados al momento de detectar las avalanchas (verdaderos positivos), el algoritmo KNN tiene mayores errores de falsos positivos, resultando de esta forma en que el algoritmo KNN

tenga una menor efectividad. La mayor efectividad del algoritmo SVM en tareas de clasificación ya se había visto en la literatura [42] y se pudo comprobar en este trabajo.

Con respecto a la comparación de los modelos SVM, se observa que el Modelo SVM 2 es más óptimo, esto ya se había previsto en la sección 5.2, donde se explicó que no necesariamente los modelos entrenados con los hiperparámetros que resulten en el modelo con la mejor efectividad (*accuracy*) teórica son mejores, también es necesario tener en cuenta que estos parámetros no sean valores muy grandes o muy pequeños. Teniendo el Modelo SVM 2 los hiperparámetros más balanceados, se esperaba que tuviera mejores resultados en las pruebas en campo.

Finalmente, habiendo definido el criterio de detección, se concluye que el mejor modelo, el Modelo SVM 2, con una efectividad del 86.46 %, tiene resultados definitivamente mejores a otros sistemas de detección con infrasonido implementados en la literatura que no utilizan *Machine Learning* [9][43], teniendo además como ventaja de que el sistema implementado en este trabajo utiliza un solo sensor en lugar de un arreglo de sensores.

SISTEMA	Tipos de sensores	Cantidad de sensores	Método utilizado en la detección	Magnitud de avalanchas analizadas	Efectividad Sistema (Accuracy)
Sistema de Detección del presente trabajo de Tesis	**Infrasonido**	1	Algoritmos de Machine Learning	Magnitudes variadas (pequeñas, medias y fuertes)	0.8646617
Sitema IDA® [12]	**Infrasonido**	4	Algoritmos de detección basados en umbrales definidos	Fuertes	0.9
				Medias	0.19
				Pequeñas	0
Sistema de detección (v1 algorithm & Manual) [46]	**Infrasonido**	4	Algoritmos de detección basados en umbrales definidos	Magnitudes variadas (pequeñas, medias y fuertes)	0.74

Tabla 12: Comparación de la efectividad de distintos sistemas de la literatura con el sistema implementado en el presente trabajo.

Fuente: Propia

VI. Costo de implementación

Para calcular el costo se consideraron los precios de los equipos, sensores y demás accesorios utilizados en el desarrollo del presente sistema, no solo en lo que respecta en la implementación sino también en algunas pruebas que se realizaron. Además se ha considerado las horas hombre que fueron necesarias para la culminación de este trabajo y también los costos de materiales de escritorio.

Cabe añadir que en lo que respecta a software, se ha utilizado solo software libre y/o gratuito, por lo que ninguna licencia ni ningún costo ha sido generado en este aspecto.

Cantidad	Equipos y Accesorios	Precio S/.
1	Sensor de infrasonido Raspberry Boom	1462.00
1	Micro ordenador Raspberry Pi 3 modelo B+	650.00
1	Cable de red ethernet cat6 (2 metros)	12.00
1	Hub de 4 puertos USB 3.0 (con cable de 30 cm)	20.00
1	Adaptador USB Wifi Realtek RTL8192EU con antena (802.11n)	45.00
1	Cable USB a micro USB (2 metros)	15.00
2	Powerbank Xiaomi Redmi 2000 mAh	220.00
1	Caja de madera Balsa	40.00
1	Servicio de barnizado	5.00
Precio Total Equipos y Accesorios		**S/. 2469.00**

Tabla 13: Costo de los equipos y accesorios utilizados en la implementación del sistema.

Fuente: Propia

Cantidad	Costo Horas-hombre	Tiempo de trabajo	Horas	Remuneración S/.
1	Bachiller (Tesista)	11 meses	5280	10 325.00
1	Ing. Electrónico (Asesor)	11 meses	220	0.00
1	Ing. de Sistemas (Co-asesor)	11 meses	350	0.00
Costo Total Horas-hombre				**S/. 10 325.00**

Tabla 14: Costo en el personal que participaron en la elaboración del presente trabajo.

Fuente: Propia

EQUIPOS Y ACCESORIOS	S/. 2469.00
MATERIALES DE ESCRITORIO	S/. 300.00
COSTO HORAS-HOMBRE	S/. 10 325.00
COSTOS DE IMPORTACIÓN	S/. 550.00
OTROS	S/. 200.00
COSTO TOTAL	**S/. 13 844.00**

Tabla 15: Resumen del costo utilizado en el análisis, diseño e implementación del sistema.

Fuente: Propia

Conclusiones

- Se diseñó e implementó exitosamente el sistema de detección de avalanchas en la laguna Palcacocha. Este sistema trabaja en tiempo real y puede ser monitoreado mediante una aplicación informática de visualización de data.

- Se implementó el sistema de monitoreo en tiempo real, en la zona próxima a la laguna existe un sistema de monitoreo del INAIGEM, el cual incluye un *access point* que provee de internet WiFi en la zona; el módulo sensor del presente trabajo utiliza esta conexión a la red para conectarse con nuestro sistema remoto ubicado en la ciudad de Huaraz.

- Se logró recolectar una gran cantidad de datos y así mismo se organizó la información en un dataset, este último esta compuesto de 105 eventos de avalancha y 115 eventos de ruido; siendo la duración de todas las muestras de 10.24 segundos.

- Se logró desarrollar el modelo de *Machine Learning* más óptimo (modelo SVM 2), dicho modelo fue entrenado bajo los siguientes criterios: se utilizó un dataset del que se extrajo 9 características a cada muestra, dichas características se definieron en el presente trabajo; se utilizó el algoritmo de *Support Vector Machine*; finalmente, los hiperparámetros utilizados fueron $C = 15$ y $gamma = 0.0008$. La efectividad (*accuracy*) teórica hallada para este modelo es de 92.27 %.

- Se implementó el análisis en tiempo real para el sistema, de forma que se analiza cada 2 segundos una señal de infrasonido con una duración de 10.24 segundos, tal que la velocidad de clasificación es de una presencia o ausencia de avalancha cada 2 segundos.

- Se realizaron pruebas piloto en la laguna Palcacocha para hallar la efectividad del modelo más óptimo en tiempo real. Se encontró que de 133 eventos detectados por el sistema, el modelo clasificó correctamente 23 avalanchas y 92 eventos de ruido, mientras que clasificó erróneamente 2 avalanchas y 16 eventos de ruido. En este sentido, la efectividad del modelo según las pruebas realizadas en campo es de 86.47 %.

- El uso de *Machine Learning* mejora la efectividad en comparación a otros sistemas de detección de infrasonido que no lo utilizan, tal como se aprecia en la tabla 15. Además el uso de *Machine Learning* permitiría obtener mejoras aún más significativas en futuros proyectos relacionados con la detección de avalanchas, ya que se puede seguir recolectando más data para posteriormente reentrenar el modelo y obtener resultados más precisos.

Recomendaciones

- Al realizar pruebas en el *Raspberry Pi*, si se desea apagarlo se debe hacerlo de forma correcta mediante un comando por consola, quitar la energía directamente puede hacer que la información en la tarjeta SD se corrompa, perdiendo de esta forma las mediciones de infrasonido almacenadas en ella.

- Si bien se utilizó una caja de madera balsa para la atenuación de ruido en este trabajo, no es estrictamente necesario que sea así, si se desea utilizar otros materiales, se puede realizar las pruebas y observar los resultados, sin embargo se recomienda usar materiales que tengan sustento en la literatura, utilizar ciertos tipos de materiales pueden hacer que se bloqueen de forma muy agresiva las señales de infrasonido.

- El método de atenuación de ruido por medio de *Windshields* cumple su trabajo, sin embargo, cuando la velocidad del viento es muy grande se sigue presentando ruidos muy notorios en las mediciones de infrasonido, por ello de ser posible, se recomienda utilizar arreglos de sensores, con los cuáles se pueden realizar eliminación de ruido mediante distintos métodos como por ejemplo la “conformación de haces” o “beamforming”, en este trabajo se utilizó un solo sensor debido a limitaciones de presupuesto.

- Si bien se pudo generar un dataset óptimo, para trabajos futuros, se recomienda ampliar la recolección de datos a la mayor cantidad de datos posibles en distintos nevados y bajo distintas circunstancias (bajo distintos tipos de ruidos y a distintos niveles).

Referencias Bibliográficas

[1] S. Allen, H. Frey, and C. G. Huggel, "Assessment of glacier and permafrost hazards in mountain regions–technical guidance document," *Standing Group on Glacier and Permafrost Hazards in Mountains (GAPHAZ) of the International Association of Cryospheric Sciences (IACS) and the International Permafrost Association (IPA): Zurich, Switzerland*, 2017.

[2] S. A. Wegner, "Lo que el agua se llevó: Consecuencias y lecciones del aluvión de huaraz de 1941," Lima: MINAM, 2014. 83 p., 2014.

[3] M. J. Rubin, T. Camp, A. Van Herwijnen, and J. Schweizer, "Automatically detecting avalanche events in passive seismic data," in *2012 11th International Conference on Machine Learning and Applications*, vol. 1, pp. 13–20, IEEE, 2012.

[4] P. Gauer, M. Kern, K. Kristensen, K. Lied, L. Rammer, and H. Schreiber, "On pulsed doppler radar measurements of avalanches and their implication to avalanche dynamics," *Cold Regions Science and Technology*, vol. 50, no. 1-3, pp. 55–71, 2007.

[5] T. Humstad, Ø. Søderblom, G. Ulivieri, S. Langeland, and H. Dahle, "Infrasound detection of avalanches in grasdalen and indreeidsdalen, norway," in *Proceedings ISSW*, pp. 621–627, 2016.

[6] S. Wahlen, L. M. S. Wyssen, and B. Arnold, "Automatic people detection in avalanche-controlled terrain during all-weather conditions," in *Conference: International Snow Science Workshop*, 2018.

[7] W. Steinkogler, S. Langeland, and C. Vera, "Operational avalanche detection systems: Experiences, physical limitations and user needs," in *7th Canadian Geohazards Conference, Canmore, Canada*, 2018.

[8] L. Meier and D. Lussi, "Remote detection of snow avalanches in switzerland using infrasound, doppler radars and geophones," in *2010 International Snow Science Workshop*, pp. 7–12, 2010.

[9] S. Mayer, A. van Herwijnen, G. Ulivieri, and J. Schweizer, "Evaluating the performance of an operational infrasound avalanche detection system at three locations

in the swiss alps during two winter seasons," *Cold regions science and technology*, vol. 173, p. 102962, 2020.

[10] J. B. Johnson, H. P. Marshall, S. M. Loo, B. Nalli, M. Saurer, S. Havens, J. Colton, and J. F. Anderson, "Detection and tracking of snow avalanches in little cottonwood canyon, utah using multiple small-aperture infrasound arrays," in *Proceedings ISSW*, pp. 616–620, 2018.

[11] G. Ulivieri, E. Marchetti, M. Ripepe, I. Chiambretti, and V. Segor, "Infrasonic monitoring of snow avalanches in the alps," in *Proceedings of international snow science workshop*, pp. 723–728, 2012.

[12] A. Schimmel, J. Hübl, R. Koschuch, and I. Reiweger, "Automatic detection of avalanches: evaluation of three different approaches," *Natural Hazards*, vol. 87, no. 1, pp. 83–102, 2017.

[13] E. D. Scott, C. T. Hayward, R. F. Kubichek, J. C. Hamann, J. W. Pierre, B. Comey, and T. Mendenhall, "Single and multiple sensor identification of avalanche-generated infrasound," *Cold Regions Science and Technology*, vol. 47, no. 1-2, pp. 159–170, 2007.

[14] T. Thüring, M. Schoch, A. van Herwijnen, and J. Schweizer, "Robust snow avalanche detection using supervised machine learning with infrasonic sensor arrays," *Cold Regions Science and Technology*, vol. 111, pp. 60–66, 2015.

[15] V. Adam, V. Chritin, M. Rossi, and E. Van Lancker, "Infrasonic monitoring of snow-avalanche activity: what do we know and where do we go from here?," *Annals of Glaciology*, vol. 26, pp. 324–328, 1998.

[16] V. Chritin, M. Rossi, and R. Bolognesi, "Acoustic detection system for operational avalanche forecasting," in *Proc. of International Snow and Science workshop*, no. CONF, 1996.

[17] E. Marchetti, M. Ripepe, G. Ulivieri, and A. Kogelnig, "Infrasound array criteria for automatic detection and front velocity estimation of snow avalanches: towards a real-time early-warning system," *Natural Hazards and Earth System Sciences*, vol. 15, no. 11, pp. 2545–2555, 2015.

[18] Q. A. Shams, A. J. Zuckerwar, and B. S. Sealey, "Compact nonporous windscreen for infrasonic measurements," *The Journal of the Acoustical Society of America*, vol. 118, no. 3, pp. 1335–1340, 2005.

[19] B. Alcoverro, "The design and performance of infrasound noise-reducing pipe arrays," in *Handbook of Signal Processing in Acoustics*, pp. 1473–1486, Springer, 2008.

[20] T. L. Szabo, "Time domain wave equations for lossy media obeying a frequency power law," *The Journal of the Acoustical Society of America*, vol. 96, no. 1, pp. 491–500, 1994.

[21] P. Gauer, D. Issler, K. Lied, K. Kristensen, and F. Sandersen, "On snow avalanche flow regimes: Inferences from observations and measurements," in *Proceedings Whistler 2008 International Snow Science Workshop September 21-27, 2008*, p. 717, 2008.

[22] D. M. McClung, "Effects of triggering mechanism on snow avalanche slope angles and slab depths from field data," *Natural hazards*, vol. 69, no. 3, pp. 1721–1731, 2013.

[23] D. McClung and P. A. Schaerer, *The avalanche handbook*. The Mountaineers Books, 2006.

[24] R. A. Sommerfeld, "Preliminary observations of acoustic emissions preceding avalanches," *Journal of Glaciology*, vol. 19, no. 81, pp. 399–409, 1977.

[25] R. Sommerfeld and H. Gubler, "Snow avalanches and acoustic emissions," *Annals of Glaciology*, vol. 4, pp. 271–276, 1983.

[26] A. Bedard, "Detection of avalanches using atmospheric infrasound," in *Proceedings of the Western Snow Conference, edited by: Shafer, B., Western Snow Conference*, pp. 52–58, 1989.

[27] G. Ulivieri, E. Marchetti, M. Ripepe, I. Chiambretti, G. De Rosa, and V. Segor, "Monitoring snow avalanches in northwestern italian alps using an infrasound array," *Cold Regions Science and Technology*, vol. 69, no. 2-3, pp. 177–183, 2011.

[28] A. Kogelnig, "Development of acoustic monitoring for alpine mass movements," Master's thesis, 2012.

[29] O. Marcillo, J. B. Johnson, and D. Hart, "Implementation, characterization, and evaluation of an inexpensive low-power low-noise infrasound sensor based on a micromachined differential pressure transducer and a mechanical filter," *Journal of Atmospheric and Oceanic Technology*, vol. 29, no. 9, pp. 1275–1284, 2012.

[30] T. Sainburg, M. Thielk, and T. Q. Gentner, "Finding, visualizing, and quantifying latent structure across diverse animal vocal repertoires," *PLoS computational biology*, vol. 16, no. 10, p. e1008228, 2020.

[31] T. Sainburg, "timsainb/noisereduce: v1.0." https://doi.org/10.5281/zenodo.3243139, June 2019.

[32] T. Bäckström, "Comparison of windowing in speech and audio coding," in *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 1–4, IEEE, 2013.

[33] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. .ºReilly Media, Inc.", 2022.

[34] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, *et al.*, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[35] M. Beyreuther, R. Barsch, L. Krischer, T. Megies, Y. Behr, and J. Wassermann, "Obspy: A python toolbox for seismology," *Seismological Research Letters*, vol. 81, no. 3, pp. 530–533, 2010.

[36] Raspberry Shake S.A., "Documentación para los productos de la marca Raspberry Shake." `https://manual.raspberryshake.org/index.html`, 2021. Accessed: 2022-08-10.

[37] Obspy, "ObsPy Documentation (v1.3.0)." `https://docs.obspy.org/`, 2010. Accessed: 2022-08-15.

[38] InfluxData, "InfluxDB Documentation (v2.3)." `https://docs.influxdata.com/influxdb/v2.3/`, 2022. Accessed: 2022-08-15.

[39] InfluxData, "Kapacitor Documentation (v1.6)." https://docs.influxdata.com/kapacitor/v1.6/, 2022. Accessed: 2022-08-15.

[40] Grafana Labs, "Grafana Documentation (v9.1.x)." https://grafana.com/docs/grafana/latest/, 2022. Accessed: 2022-08-15.

[41] M. Barandas, D. Folgado, L. Fernandes, S. Santos, M. Abreu, P. Bota, H. Liu, T. Schultz, and H. Gamboa, "Tsfel: Time series feature extraction library," *SoftwareX*, vol. 11, p. 100456, 2020.

[42] S. Prabavathy, V. Rathikarani, and P. Dhanalakshmi, "Classification of musical instruments using svm and knn," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 7, pp. 1186–1190, 2020.

[43] J. Hendrikx, L. Dreier, G. Ulivieri, J. Sanderson, A. Jones, and W. Steinkogler, "Evaluation of an infrasound detection system for avalanches in rogers pass, canada," in *Proceedings ISSW*, pp. 171–175, 2018.

Anexos

Anexo 01: Especificaciones del Raspberry Pi 3 Modelo B+

Procesador	Broadcom BCM2837B0 Cortex-A53 64-bit SoC @ 1.4GHz
Memoria	1GB LPDDR2 SDRAM
Conectividad	2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN Bluetooth 4.2, BLE Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps) 4 × USB 2.0 ports
Acceso	Extended 40-pin GPIO header
Video y sonido	1 × full size HDMI MIPI DSI display port MIPI CSI camera port 4 pole stereo output and composite video port
Multimedia	H.264, MPEG-4 decode (1080p30) H.264 encode (1080p30) OpenGL ES 1.1, 2.0 graphics
SD card support	Micro SD format for loading operating system and data storage
Input Power	5 V/2.5 A DC via micro USB connector 5 V DC via GPIO header Power over Ethernet (PoE)–enabled (requires separate PoE HAT)

Anexo 02: Especificaciones del sensor de infrasonido Raspberry Boom

Parámetro	**Valor**
Tipo de sensor	MEMS temperature compensated differential pressure transducer
Muestras por segundo	100
Velocidad de transmisión de paquetes de datos	Data packets shipped across serial port at a rate of 4 packets/ second (250 ms/ packet)
Ancho de Banda (estimación)	-3dB points at 1 Hertz (1 seconds) to 44 Hertz (for 1s mechanical filter, default). -3dB points at 0.08 Hertz (13 seconds) to 44 Hertz (for 20s mechanical filter). Rolloff past low frequency corners: 2 poles or 40dB/decade
Polos (estimación, radianes por segundo)	There is a hardware single-pole high-pass filter with a -3 dB point around 0.05 Hz. With 1s mechanical filter attached: -0.312 (20 seconds, single pole high pass filter, from hardware) -6.289 (1 Hz, single pole high pass filter, from mechanical filter) With 20s mechanical filter attached: -0.312 (20 seconds, single pole high-pass filter, from hardware) -0.312 (20 seconds, single pole high pass filter, from mechanical filter)
Zeros (estimación, radianes por segundo)	0,0
Sensibilidad (estimación)	56,000 counts/ Pascal +/- 10 % precision
Nivel de corte (estimación)	+/- 8,388,608 counts (24-bits) 0.5 inches of water, corresponding to +/- 125 Pa
Digitalizador	24-bit ADC Sigma-Delta ΣΔ
Rango dinámico	144 dB (24 bits)
Bits efectivos (estimación)	21 bits (126 dB) from 1 to 20 Hz @ 100 sps (for the entire analog to digital hardware chain)
Banda de error	1 %
Linealidad de la medición de presión (incluida en la medición de la banda de error total)	<0.5 %
Calibración de la ganancia	Automatic
Opciones para el filtro mecánico pasa altos	1s, 20s (all units ship with both)
Temperatura operacional del sensor	Compensated operating range: 0 to 50 C Max. operating range: -25 to 85 C (though the rest of the electronics are limited to 0-60C)

Anexo 03: Archivo JSON con características a extraer por la librería TSFEL

```
{
  "spectral": {
    "Spectral centroid": {
      "complexity": "linear",
      "function": "tsfel.spectral_centroid",
      "parameters": {
        "fs": 100
      },
      "n_features": 1,
      "use": "yes"
    },
    "Spectral kurtosis": {
      "complexity": "linear",
      "function": "tsfel.spectral_kurtosis",
      "parameters": {
        "fs": 100
      },
      "n_features": 1,
      "use": "yes"
    },
    "Spectral positive turning points": {
      "complexity": "log",
      "function": "tsfel.spectral_positive_turning",
      "parameters": {
        "fs": 100
      },
      "n_features": 1,
      "use": "yes"
    },
    "Spectral roll-off": {
      "complexity": "log",
      "function": "tsfel.spectral_roll_off",
      "parameters": {
        "fs": 100
      },
      "n_features": 1,
      "use": "yes"
    },
    "Spectral skewness": {
      "complexity": "linear",
      "function": "tsfel.spectral_skewness",
      "parameters": {
        "fs": 100
      },
      "n_features": 1,
      "use": "yes"
    },
    "Spectral slope": {
      "complexity": "log",
      "function": "tsfel.spectral_slope",
      "parameters": {
        "fs": 100
      },
      "n_features": 1,
      "use": "yes"
    },
    "Spectral spread": {
      "complexity": "linear",
      "function": "tsfel.spectral_spread",
      "parameters": {
```

```
            "fs": 100
        },
        "n_features": 1,
        "use": "yes"
    }
  },
  "temporal": {
    "Zero crossing rate": {
      "complexity": "constant",
      "function": "tsfel.zero_cross",
      "parameters": "",
      "n_features": 1,
      "use": "yes"
    }
  }
}
```

Anexo 04: Archivo de configuración de Kapacitor

```
hostname = "DESKTOP-33GQDTE"
data_dir = "C:\\Program Files\\InfluxData\\kapacitor\\kapacitor-1.6.4-1\\data_dir"
skip-config-overrides = true
default-retention-policy = ""

[alert]
  persist-topics = true
  topic-buffer-length = 5000

[http]
  bind-address = ":9092"
  auth-enabled = false
  log-enabled = true
  write-tracing = false
  pprof-enabled = false
  https-enabled = false
  https-certificate = "/etc/ssl/kapacitor.pem"
  https-private-key = ""
  shutdown-timeout = "10s"
  shared-secret = ""

[replay]
  dir = "C:\\Users\\HP\\.kapacitor\\replay"

[storage]
  boltdb = '''C:\\Program Files\\InfluxData\\kapacitor\\kapacitor-1.6.4-1\\data_dir\\
  kapacitor.db'''

[task]
  dir = "C:\\Program Files\\InfluxData\\kapacitor\\kapacitor-1.6.4-1\\data_dir\\tasks"
  snapshot-interval = "1m0s"

[load]
  enabled = false
  dir = "C:\\Program Files\\InfluxData\\kapacitor\\kapacitor-1.6.4-1\\data_dir\\load"

[[influxdb]]
  enabled = true
  name = "default"
  default = true
  urls = ["http://127.0.0.1:8086"]
  username = ""
  password = ""
  token = ""
  http-shared-secret = false
  ssl-ca = ""
  ssl-cert = ""
  ssl-key = ""
  insecure-skip-verify = true
  timeout = "0s"
  disable-subscriptions = true
  subscription-protocol = "http"
  subscription-mode = "cluster"
  kapacitor-hostname = ""
  http-port = 0
  udp-bind = ""
  udp-buffer = 1000
  udp-read-buffer = 0
  startup-timeout = "5m0s"
  subscriptions-sync-interval = "1m0s"
  subscription-path = ""
  compression = "gzip"
  [influxdb.excluded-subscriptions]
```

```
64      _kapacitor = ["autogen"]
65
66  [logging]
67    file = "STDERR"
68    level = "DEBUG"
69
70  [config-override]
71    enabled = true
72
73  [udf]
74  [udf.functions]
75    [udf.functions.deteccionAvalancha]
76      prog = "C:\\Users\\HP\\AppData\\Local\\Programs\\Python\\Python310\\python.exe"
77      args = [ "-u", '''C:\\Program Files\\InfluxData\\kapacitor\\kapacitor-1.6.4-1\\
78      udf_test.py''']
79      timeout = "15s"
80      [udf.functions.deteccionAvalancha.env]
81        PYTHONPATH = '''C:\\Program Files\\InfluxData\\kapacitor\\kapacitor-1.6.4-1\\
82        kapacitor\\udf\\agent\\py'''
```

Anexo 05: Implementación en python del agente UDF de Kapacitor

```
from kapacitor.udf.agent import Agent, Handler
from kapacitor.udf import udf_pb2
import logging
import json

from sqlalchemy import false

from codigo3 import modulo_pre_procesamiento
from codigo5 import modulo_extraccion_caracteristicas
from codigo15 import clasificacion_eventos

# Importacion Muestra de ruido de 20 segundos
from codigo1 import obtener_señal_de_archivo_miniSEED
señal_ruido = obtener_señal_de_archivo_miniSEED("2022-05-09T05:07:09", 20,
                                                "20220509_1001")

# Importacion del modelo optimo para realizar la clasificacion
from codigo9 import cargar_modelo
path = ""  #el path es "" debido a que el modelo se encuentra en la misma carpeta
modelo= cargar_modelo("model_KNN_10.24_final.pkl", path)

logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s %(levelname)s:%(name)s: %(message)s')
logger = logging.getLogger()

class GeoTestHandler(Handler):
    def __init__(self, agent):
        """Constructor"""
        logger.info('__init__ trigger')
        self._agent = agent
        self._field = ' '
        #self._size = True
        self._points = []
        #self._batch = None
        self._state = {}

    def info(self):
        """info: Define what your UDF wants and what will it provide in the end"""
        logger.info('info trigger')
        response = udf_pb2.Response()
        response.info.wants = udf_pb2.BATCH
        response.info.provides = udf_pb2.STREAM
        response.info.options['field'].valueTypes.append(udf_pb2.STRING)
        return response

    def init(self, init_req):
        """init: Define what your UDF expects as parameters when parsing the
        TICKScript"""
        logger.info('INIT trigger')
        for opt in init_req.options:
            if opt.name == 'field':
                self._field = opt.values[0].stringValue
        success = True
        msg = ''
        if self._field == '':
            success = False
            msg = 'must provide field name'
        response = udf_pb2.Response()
        response.init.success = success
        response.init.error = msg.encode()
        return response

    def begin_batch(self, begin_req):
```

```
64          """begin_batch: Do something at the beginning of the batch"""
65          logger.info('begin_batch trigger')
66
67      def snapshot(self):
68          """snapshot: take a snapshot of the current data, if the task stops for
69          some reason """
70          data = {}
71          for group, state in self._state.items():
72              data[group] = state.snapshot()
73          response = udf_pb2.Response()
74          response.snapshot.snapshot = json.dumps(data).encode()
75          return response
76
77      def point(self, point):
78          """point: process each point within the batch"""
79          #self._batch.update(point.fieldsDouble[self._field])
80          response = udf_pb2.Response()
81          self._points.append(point.fieldsDouble[self._field])
82
83          # primero se extrae una señal de los últimos 10.24 segundos
84          # en tiempo real que corresponden a 1024 puntos.
85          if len(self._points) == 1024:
86              logger.info(len(self._points))
87
88              # Se filtra la señal
89              señal_filtrada= modulo_pre_procesamiento(self._points, señal_ruido, 100)
90
91              # Se extrae caracteristicas de señal filtrada
92              vector_caracteristicas= modulo_extraccion_caracteristicas(señal_filtrada,
93                                                                        100,
94                                                                        "features.json",
95                                                                        256,128)
96
97              # Se obtiene el resultado de la claisficacion a partir del modelo
98              # de machine learning optimo. El resultado puede ser un 1 o un 0.
99              resultado_deteccion= clasificacion_eventos(vector_caracteristicas, modelo)
100
101             # Captar la marca de tiempo del último punto del
102             # fragmento de análisis y guardarlo en la
103             #variable response
104             response.point.name = point.name
105             response.point.time = point.time
106             response.point.group = point.group
107             response.point.tags.update(point.tags)
108             # añadir el resultado de la clasificación a la variable response,
109             # el campo con el que se guarda el resultado de la clasificación
110             # se llamará 'detecciones'.
111             response.point.fieldsDouble['detecciones'] = resultado_deteccion
112             # escribir en la base de datos la variable response
113             self._agent.write_response(response)
114             self._points=[]
115
116     def end_batch(self, batch_meta):
117         """end_batch: do something at the end of the batch"""
118         #self._size = True
119         #logger.info(self._size)
120         logger.info(len(self._points))
121         logger.info('end_batch')
122
123
124 if __name__ == '__main__':
125     # Create an agent
126     agent = Agent()
127
128     # Create a handler and pass it an agent so it can write points
129     h = GeoTestHandler(agent)
130
```

```
131         # Set the handler on the agent
132         agent.handler = h
133 
134         # anything printed to STDERR from a UDF process gets captured into the logs
135         logger.info("Starting agent for GeoTestHandler")
136         agent.start()
137         agent.wait()
138         logger.info("Agent finished")
```

Anexo 06: Convenio para el desarrollo de tesis con el INAIGEM

PERÚ | Ministerio del Ambiente | Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña

"Decenio de la Igualdad de Oportunidades para Mujeres y Hombres"
"Año del Bicentenario del Perú: 200 años de Independencia"

CONVENIO DE TESIS

CONVENIO DE ASISTENCIA Y SUBSIDIO DE TESIS DE INVESTIGACIÓN CIENTÍFICA Y ACADÉMICA

Conste por el presente documento el Convenio de Asistencia y Subsidio de Tesis de investigación científica y académica que celebran, de una parte, el **INSTITUTO NACIONAL DE INVESTIGACIÓN EN GLACIARES Y ECOSISTEMAS DE MONTAÑA**, en adelante **INAIGEM**, con RUC N° 20600404262, con domicilio legal en Jirón Juan Bautista Mejía N° 687, distrito y provincia de Huaraz, departamento de Ancash, representado por el Abg. Rocco Romero Grados, en calidad de Jefe de la Oficina de Administración, designado con Resolución de Presidencia Ejecutiva N° 024-2021-INAIGEM/PE, identificado con DNI N° 07521922; y de la otra parte, el señor **Benites Condori Cristian Adriel**, identificado con DNI N° 73049941, en adelante **EL TESISTA**, con domicilio en Av. Tullumayu N° 123, Distrito de Wanchaq, Provincia de Cusco Departamento de Cusco, en los términos y condiciones siguientes:

CLAUSULA PRIMERA. - ANTECEDENTES

El **INAIGEM** y **EL TESISTA** han coordinado las acciones vinculadas con la suscripción del presente Convenio de Asistencia y Subsidio de Tesis, a fin de facilitar la investigación científica y académica sobre temas de interés mutuo, en concordancia con las metas y objetivos comunes, en beneficio de la población.

CLAUSULA SEGUNDA. - PARTES

2.1. El **INAIGEM** es un organismo técnico especializado creado mediante Ley N° 30286, adscrito al Ministerio del Ambiente, con personería jurídica de derecho público, con competencia nacional y autonomía administrativa, funcional, técnica, económica y financiera. Es la máxima autoridad en la investigación científica de los glaciares y ecosistemas de montaña y tiene como función principal fomentar y expandir la investigación científica y tecnológica en el ámbito de los glaciares y los ecosistemas de montaña.

2.2. **EL TESISTA**, es una persona natural que está en condiciones de desarrollar investigaciones científicas y académicas orientadas a la formación y desarrollo de una Tesis que contribuya a los fines de investigación científica aplicada del **INAIGEM**. **EL TESISTA** cumple con los requisitos indicados en la Directiva N°002-2021-INAIGEM/GG.

CLAUSULA TERCERA. - MARCO LEGAL

- Constitución Política del Perú.
- Ley N° 28611, Ley General del Ambiente.
- Ley N° 29158, Ley Orgánica del Poder Ejecutivo.
- Ley N° 30286 que crea el Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña–INAIGEM.
- TUO de la Ley N° 27444, Ley del Procedimiento Administrativo General.
- Ley N° 30220, Ley Universitaria.
- Decreto Supremo N°012-2009-MINAM establece la Política Nacional de Ambiente.
- Decreto Supremo N°011-2015, establece la Estrategia Nacional ante el Cambio Climático.

- Decreto Supremo N°005-2020-MINAM que aprueba el Reglamento de Organización y Funciones del INAIGEM.
- Directiva N° 002-2021-INAIGEM/GG, "Lineamientos del Programa interno de Tesistas en el Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña-INAIGEM"

CLAUSULA CUARTA. - OBJETO DEL CONVENIO

El presente Convenio tiene por objeto prestar orientación asistida y brindar subvención económica a **EL TESISTA** y obtener la contribución de esta en el desarrollo de investigaciones de interés del **INAIGEM**, que contribuyan con los fines y objetivos de la entidad, en beneficio de la población.

CLÁUSULA QUINTA. - COMPROMISOS

Las partes asumen los siguientes compromisos:

5.1. El **INAIGEM**, acepta colaborar con la investigación de **EL TESISTA**, en coordinación con la universidad. Para ello el co-asesor designado por el **INAIGEM** coordinará con el asesor de tesis designado por dicho centro.

5.2. El **INAIGEM** brindará apoyo a los **TESISTAS**, cuya residencia no se encuentre en las sedes del **INAIGEM** proveyendo: Listado de alojamientos, recojo de la estación de bus y coordinación de su primer alojamiento, entre otros.

5.3. El **INAIGEM** y **EL TESISTA**, para efecto de la elaboración de la tesis se sujetarán las reglas administrativas contenidas en la Directiva N° 002-2021-INAIGEM/GG: "Lineamientos del Programa de Tesistas en el Instituto Nacional de Investigación en Glaciares y Ecosistemas de Montaña-INAIGEM", la misma que forma parte integrante del presente Convenio.

5.4. **EL TESISTA** llevará un cuaderno de campo y laboratorio que es de propiedad del **INAIGEM.** En el mismo anotará sus actividades, llevará el registro de datos y resultados, y al término de su tesis, entregará un ejemplar a su co-asesor.

5.5. **EL TESISTA** está obligado a proporcionar toda la documentación (resolución del consejo de facultad o resolución que aprueba el proyecto de tesis) que lo acredita como tesista de la universidad correspondiente, precisando la facultad/dependencia que da respaldo académico a su tesis.

5.6. **EL TESISTA** se compromete a no tener otras ocupaciones y dedicarse en exclusividad a la elaboración de la tesis.

5.7. La información general del/EL TESISTA:
a) Universidad: Universidad Nacional de San Antonio de Abad del Cusco
b) Formación Profesional: Ingeniería Electrónica
c) Condición de EL TESISTA que suscribe el convenio: Bachiller (x) /Titulado ( ) / Maestro o Magister ( ) / Doctor ( )
d) Área o dependencia del **INAIGEM**, donde se realizará la tesis: **Dirección de Información y Gestión de Conocimiento.**

5.8. Nombre del co-asesor: Robert Alfredo Alvarado Lugo

5.9. Monto de la subvención a entregar al **TESISTA** al final de cada mes (Alimentación / hospedaje / transporte): S/ 930.00 mensuales.

5.10. **EL TESISTA** deberá entregar un informe de avances trimestral que será aprobado por el co-asesor y director de línea respectiva, previo a la entrega de la subvención mensual.

5.11. **EL TESISTA** está obligada a cumplir con el calendario de actividades establecido al inicio del trabajo, junto con el co-asesor del **INAIGEM**. En caso que estas fechas no se respeten deberá de presentar una justificación en el informe de avances trimestral, que sea aprobado por el co-asesor y director de línea respectivo.

5.12. **EL TESISTA** está obligada a prestar asistencia y/o apoyo en actividades del INAIGEM vinculadas a su investigación cuando le sea solicitado.

5.13. Para dar por termino el presente convenio y otorgar la última subvención, EL TESISTA deberán de entregar:

a) Acta de sustentación de tesis, y es deseable que entregue el diploma de grado o título profesional registrado en la **SUNEDU**.

b) Tesis en formato digital entregada en la biblioteca del **INAIGEM,** donde se reconozca el financiamiento del **INAIGEM** y la afiliación del **INAIGEM** y del co-asesor.

c) Artículo enviado a una revista indexada y revisada por pares, obligatorio para tesistas de maestría y deseable para tesistas de pre grado.