INSTITUTO TECNOLÓGICO DE LA PAZ
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
MAESTRÍA EN SISTEMAS COMPUTACIONALES

CLASIFICACIÓN DE ENJAMBRES Y RÉPLICAS SÍSMICAS MEDIANTE APRENDIZAJE AUTOMÁTICO

T E S I S

QUE PARA OBTENER EL GRADO DE
MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:
ALFREDO AGUIRRE ESTRADA

DIRECTORES DE TESIS:
DR. ISRAEL MARCOS SANTILLÁN MÉNDEZ
DR. ROBERTO ORTEGA RUIZ

LA PAZ, BAJA CALIFORNIA SUR, MÉXICO, ENERO 2021.

Instituto Tecnológico de La Paz
Representante Institucional ante PRODEP

La Paz, B.C.S., 20/enero/2021

DEPI_MSC/018/2021

ASUNTO: Autorización de impresión

C. ALFREDO AGUIRRE ESTRADA,
ESTUDIANTE DE LA MAESTRÍA EN SISTEMAS COMPUTACIONALES,
P R E S E N T E .

Con base en el dictamen de aprobación emitido por el Comité Tutorial de la Tesis denominada: **"CLASIFICACIÓN DE ENJAMBRES Y RÉPLICAS SÍSMICAS MEDIANTE APRENDIZAJE AUTOMÁTICO"**, mediante la opción de tesis (Proyectos de Investigación), entregado por usted para su análisis, le informamos que se **AUTORIZA** la impresión

A T E N T A M E N T E
Excelencia en Educación Tecnológica®

JUAN PABLO MORALES ÁLVAREZ,
JEFE DE LA DIV. DE ESTUDIOS DE POSGRADO E INV.

INSTITUTO TECNOLÓGICO DE LA PAZ
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p. Depto. de Servicios Escolares
c.c.p. Archivo.

*JPMA/icl**

Instituto Tecnológico de La Paz
División de Estudios de Posgrado e Investigación

DICTAMEN DEL COMITÉ TUTORIAL

La Paz, B.C.S., **18/ENERO/ 2021**

C. JUAN PABLO MORALES ALVAREZ,
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN,
P R E S E N T E.

Por medio del presente, enviamos a usted dictamen del Comité Tutorial de tesis para la obtención del grado de Maestro, con los siguientes datos generales:

No. de Control M19310007	Nombre ALFREDO AGUIRRE ESTRADA
Maestría en:	SISTEMAS COMPUTACIONALES
Título de la tesis: CLASIFICACIÓN DE ENJAMBRES Y RÉPLICAS SÍSMICAS MEDIANTE APRENDIZAJE AUTOMÁTICO	
DICTAMEN: Se autoriza el trabajo de investigación, en virtud de que realizó las correcciones correspondientes conforme a las observaciones planteadas por este Comité Tutorial.	

A t e n t a m e n t e .
El Comité Tutorial

DR. ROBERTO ORTEGA RUIZ

DR. SAÚL MARTÍNEZ DÍAZ

DR. ISRAEL MARCOS SANTILLÁN MÉNDEZ

c.c.p. Coordinador de la Maestría.
c.c.p. Departamento de Servicios Escolares.
c.c.p. Estudiante.

Boulevard Forjadores de B.C.S. #4720,
Col. 8 de Octubre 1ra. Sección, C.P. 23080,
La Paz, B.C.S.
Tels. (612) 121-04-24,
email: depi_paz@tecnm.mx
tecnm.mx | lapaz.tecnm.mx

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

CARTA CESIÓN DE DERECHOS

La presente se extiende en la Ciudad de La Paz, B.C.S. El día 12 del mes Enero del año 2021, el (la) que suscribe Alfredo Aguirre Estrada estudiante del Programa de **Maestría en Sistemas Computacionales** con número de control M19310007, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Israel Marcos Santillan Méndez y cede los derechos del trabajo intitulado Clasificacion de Enjambres y Replicas Sísmicas mediante Aprendizaje Automático, en forma NO EXCLUSIVA, al Tecnológico Nacional de México/Instituto Tecnológico de la Paz para su reproducción total o parcial en cualquier medio con fines académicos, científicos y culturales, así como para su publicación electrónica del texto completo para difusión y consulta.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección i2fredaguirre@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Alfredo Aguirre Estrada

Nombre y firma

Dedicatoria

Este trabajo está dedicado:

A mis padres: Tio Fifi y Tia Lucy +QEPD,
Por darme la vida y fundamentalmente por inculcarme los valores que ahora poseo,
que fueron piedra angular para poder terminar con éxito
mi carrera profesional que ahora les dedico.
Gracias.

A mis hijos: Marian Alexandra, Cesar Alfredo y Saul André Aguirre Vega,
son el motor de mi vida y fuente de mi esfuerzo.

Agradecimientos

Agradezco a Dios primeramente,
por permitirme culminar esta etapa en mi vida profesional.

Al Dr. Roberto Ortega por su valioso apoyo y orientación,
sin el cual no hubiera sido posible la culminación de la maestría.
A CICESE Unidad La Paz,
por el tiempo que me brindo para superarme profesionalmente.

A mi director de tesis, Dr. Israel Marcos Santillan Méndez,
por el tiempo, orientacion y aportaciones en la dirección de la presente tesis.
A la Maestra Iliana y Dr. Marco Antonio Castro Liera, y Dr. Saul Martinez Diaz
por su apoyo, orientación y consejos.
A todos mis maestros del posgrado del TECNM campus La Paz.
A mis compañeros de maestría,
en especial a mi compañera Viviana Flores.

A mis compañeros de CICESE Unidad La Paz:
Dra. Dana Carciumaru, Dr. Hugo Herrera y Hernando Torres,
por el apoyo y palabras de aliento.

a Mariela..
has sido la motivación y el ingrediente para alcanzar esta etapa en mi vida.

Una meta ha sido cumplida y mil palabras no bastaran para agradecerles:
su apoyo, su comprensión y sus consejos en los momentos difíciles.
Que Dios los bendiga y los guarde para siempre.

"Mientras los filósofos discuten si es posible o no

la inteligencia artificial . . .

los investigadores la construyen"

C. Frabetti

Resumen

La península de Baja California en México, es una región tectónicamente activa que presenta una intensa actividad sísmica, también es propensa al impacto de ciclones tropicales. Ambos fenómenos naturales suelen ocurrir de manera aislada, pero a principios de septiembre de 2007 ambos ocurrieron de manera casi simultánea en la ciudad de La Paz, Baja California Sur: El día 1 de septiembre de 2007 el terremoto de Mw = 6.1 (magnitud momento) en el Golfo de California, cerca de la Isla Cerralvo a unos 110 kilómetros al noreste de la ciudad de La Paz, y de manera casi simultánea, el día 4 de septiembre durante la secuencia de réplicas el huracán Henriette de categoría 1.

Este terremoto generó una gran cantidad de réplicas, debido probablemente a la presencia del Huracán Henriette, tan solo de Septiembre de 2007 a Enero de 2008 se tienen contabilizados 3,996 registros. El departamento de Sismología de CICESE unidad La Paz, ha podido localizar casi 800 réplicas, casi el 20 %, pero existe una gran cantidad de réplicas perdidas, un poco más del 80 %.
Para evitar estas pérdidas y reconocer las réplicas se propone un nuevo enfoque de búsqueda automática mediante aprendizaje automático. Se desarrollará un sistema que clasifica las réplicas mediante una Red Neuronal Artificial ANN *Artificial Neural Network*. Las ANNs tienen un uso relativamente nuevo en el reconocimiento de patrones sísmicos, representan una alternativa muy valiosa al minimizar carga de trabajo a un analista humano y presentan resultados en poco tiempo.

Se trabajarán con los eventos sísmicos iguales o mayores a 3 de magnitud coda y con el canal vertical. Se extraen las características en 2 etapas: En el preprocesamiento para preparar la señal sísmica y el procesamiento, en el cual se divide la señal sísmica en 6 segmentos, y se extraen las características en el dominio del tiempo y de la frecuencia obteniendo un vector de 120 características.

Después se definen 2 experimentos : el primero con las 120 entradas, y el segundo mediante un análisis de reducción de dimensionalidad aplicando selección y extracción de características, se reduce a 50 componentes de entrada. Mediante las librerías Keras y Talos de Python se optimiza la arquitectura de los 2 modelos de ANN y se evalúan los resultados mediante las principales tasas y métricas de clasificación para determinar el óptimo.

Abstract

The Baja California peninsula in Mexico, a tectonically active region with intense seismic activity, is also a zone subject to the impact of tropical cyclones. Both natural phenomena usually occur in isolated manner, but in early September 2007 both occurred almost simultaneously in the city of La Paz, Baja California Sur: On September 1, 2007 the earthquake of Mw = 6.1 (moment magnitude) in the Gulf of California, near Cerralvo Island about 110 kilometers northeast of the city of La Paz, and almost simultaneously, on September 4 during the sequence of aftershocks the category 1 hurricane Henriette.

This earthquake generated a large number of aftershocks, probably caused by the presence of Hurricane Henriette. From September 2007 to January 2008 only, 3,996 records have been recorded. The Seismology Department of CICESE La Paz unit, has been able to locate almost 800 aftershocks, almost 20 %, but there is a large number of missing aftershocks, a little more than 80 %.
To avoid these losses and recognize the replicas, a new automatic search approach using machine learning is proposed. A system will be developed that classifies replicas using an Artificial Neural Network (ANN). ANNs have a relatively new use in seismic pattern recognition, represent a very valuable alternative by minimizing workload to a human analyst and present results in a short time.

We will work with seismic events equal or greater than 3 coda magnitude and with the vertical channel. Features are extracted in 2 stages: In preprocessing to prepare the seismic signal and processing, in which the seismic signal is divided into 6 segments, and features are extracted in the time and frequency domain obtaining a vector of 120 features.

Then 2 experiments are defined: the first one with the 120 inputs, and the second one by means of a dimensionality reduction analysis applying feature selection and extraction, is reduced to 50 input components. Using Python Keras and Talos libraries, the architecture of the 2 ANN models is optimized and the results are evaluated using the main classification rates and metrics to determine the optimum.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN

Desde la antigüedad hasta la edad media se desconocía la causa que provocaban los sismos, se le asociaba generalmente a una explicación mítica, al movimiento de criaturas al interior de la tierra, como se explica en la figura 1.0.1.

Figura 1.0.1: La mitología en Japón aseguraba que “Namazu”, especie de inmenso bagre, hacía temblar la tierra al mover su cola.

No fue sino hasta finales del siglo XVIII cuando se da el nacimiento de la sismología como ciencia. El día 1 de noviembre de 1755, alrededor de las 9:40 am, una serie de terremotos de gran magnitud, seguido de un tsunami y una sucesión de incendios, destruyó prácticamente la ciudad de Lisboa, en Portugal, y fue sentido en toda Europa occidental. Miles de personas murieron bajo las ruinas o en las llamas, entre 60 000 y 100 000 personas (figura 1.0.2). El epicentro fue ubicado en el océano Atlántico a unos 200 Km. al suroeste de cabo San Vicente al sur de Portugal y su magnitud ha sido estimada

cerca de 9.0 en la escala de Richter. Es considerado por algunos investigadores como el primer gran desastre moderno y se le conoce como el Gran Terremoto de Lisboa [1].

Figura 1.0.2: Alegoría del terremoto de 1755 y sus consecuencias, João Glama Strobërle (1708–1792).

Posterior al evento catastrófico el primer ministro del rey José de Portugal Sabastiao José de Carvalho e Melo, mejor conocido como Marqués de Pombal convocó a jornadas coordinadas de actividades de búsqueda y rescate, también puso en marcha las etapas de reconstrucción que incluyo la incorporación de medidas para mitigar los efectos destructores de futuros terremotos y envió un cuestionario a todas las parroquias con las preguntas: ¿Cuántas réplicas se sintieron?, ¿Cuál fue la duración?, ¿Qué tipo de daños se ocasionaron?, ¿Qué ocurrió en los pozos de agua? Y ¿Se notó un comportamiento extraño en animales?

Estas preguntas todavía son utilizadas en los cuestionarios posteriores a un sismo para preparar mapas de isosistas de intensidad sísmica. Sin este tipo de información hubiera sido muy difícil para los científicos modernos analizar el Gran Terremoto de Lisboa [2].

Este evento fue el primer terremoto estudiado científicamente que tuvo efectos sobre una gran área, por lo que marcó las bases de la sismología moderna, es por ello que algunos investigadores señalan que este temblor fue el primero en generar el interés científico dando comienzo al verdadero estudio del origen de los sismos.

Hoy sabemos que son fenómenos naturales, que tienen su origen en el interior de la tierra, que generan ondas que se propagan en todas direcciones y se deben a la liberación de energía por el desplazamiento de fracturas, generalmente en las proximidades de una falla geológica. Brindan información sobre el

interior de la tierra que independiente de los daños materiales y pérdida de vidas que estos ocasionan, tiene aplicaciones en minería, petróleo, etc.

La palabra sismo es sinónimo de terremoto. La palabra sismo proviene del griego *seismos* = agitación o sacudimiento de la tierra (*-seien*= sacudir o balancear y el sufijo *-mos* para crear sustantivos a partir de verbos). Mientras que la palabra terremoto se deriva del latín *terraemotus*= movimiento de la tierra (*terra*= tierra y *motus*= movimiento).

La sismología es la ciencia que estudia los sismos y los fenómenos asociados como maremotos, tsunamis y el vulcanismo, las causas que los originan, lo que ayuda a conocer la estructura interna de la tierra.

Los objetivos de la sismología son:

- Estudiar la fuente que los produce.
- Estudiar las ondas que generan.
- Entender el medio físico que atraviesan.

La mayor parte de la investigación que realizan los sismólogos se basa en el análisis de las formas de onda registradas en sismogramas. Un sismograma es una serie temporal que consiste de una secuencia de valores de desplazamiento, velocidad o aceleración observada en el tiempo:

- Ordenada cronológicamente.
- Su comportamiento muestra patrones temporales.
- Los patrones permiten distinguir fases sísmicas.

En algunos casos los sismogramas incluyen 3 componentes para determinar el movimiento simultáneo: Este-Oeste (E), Norte-Sur (N) y Vertical (Z). Cada componente contiene información del terremoto sobre cada dirección de movimiento.

1.1. Antecedentes

La península de Baja California tiene una longitud aproximada de 1300 km y un ancho entre 45 y 240 km, está localizada en el noroeste de México, y se encuentra separada de territorio continental por medio del Golfo de California. Esta región tectónicamente presenta una intensa actividad sísmica y es

propensa a ciclones tropicales los cuales azotan la parte sur de la península con mayor frecuencia de agosto a septiembre de cada año. Los terremotos y huracanes representan dos importantes fenómenos naturales impredecibles, no siguen protocolos similares de planes de prevención de desastres. Mientras el protocolo de respuesta ante un terremoto requiere la evacuación de edificios, en presencia de huracanes, el protocolo recomienda acciones opuestas como permanecer en un sitio seguro.
A pesar de que los terremotos y huracanes suelen ocurrir de manera aislada, a principios del mes de septiembre de 2007 se presentó un evento único donde ambos fenómenos ocurrieron simultáneamente:

El terremoto de Cerralvo de magnitud momento Mw = 6.1 ocurrió en el Golfo de California el día 1 de septiembre de 2007 a unos 110 kilómetros al noreste de la ciudad de La Paz. El evento principal se ubicó en 24.957º N y -109.753º W ± 5 km. Es el mayor terremoto ocurrido en esta región en los últimos 20 años. De manera simultánea, durante la secuencia de réplicas el día 4 de septiembre de 2007 el huracán Henriette azotó la ciudad de La Paz, Baja California Sur.
Este terremoto fue inusual y genero una gran cantidad de réplicas de Septiembre de 2007 a Enero de 2008, contabilizado un total de 1475 réplicas dentro de 3,996 registros, probablemente debidas a la presencia del Huracán Henriette [3].

El CICESE unidad La Paz, ha podido localizar casi 800 réplicas usando las estaciones de NARS-Baja, pero existe un gran número de réplicas perdidas. La sismicidad localizada asi como la magnitud de las replicas se muestra en la figura 1.1.1

1.2. Descripción del problema

Para evitar las perdidas y reconocer las réplicas se requiere de un nuevo enfoque de búsqueda automatizada, por lo que se propone desarrollar un modelo de red neuronal artificial para reconocer los terremotos en las réplicas.

Área de estudio y sismicidad registrada

La información es proporcionada por el departamento de Sismología de CICESE Unidad la Paz mediante un archivo csv con los registros de la sismicidad comprendida de Agosto a Diciembre de 2007 de la región sur del golfo de California, compuesto de 3,996 registros de los cuales se tienen localizados 788 (20 %) y sin localizar 3,208 (80 %). Como se muestra en la figura 1.2.1

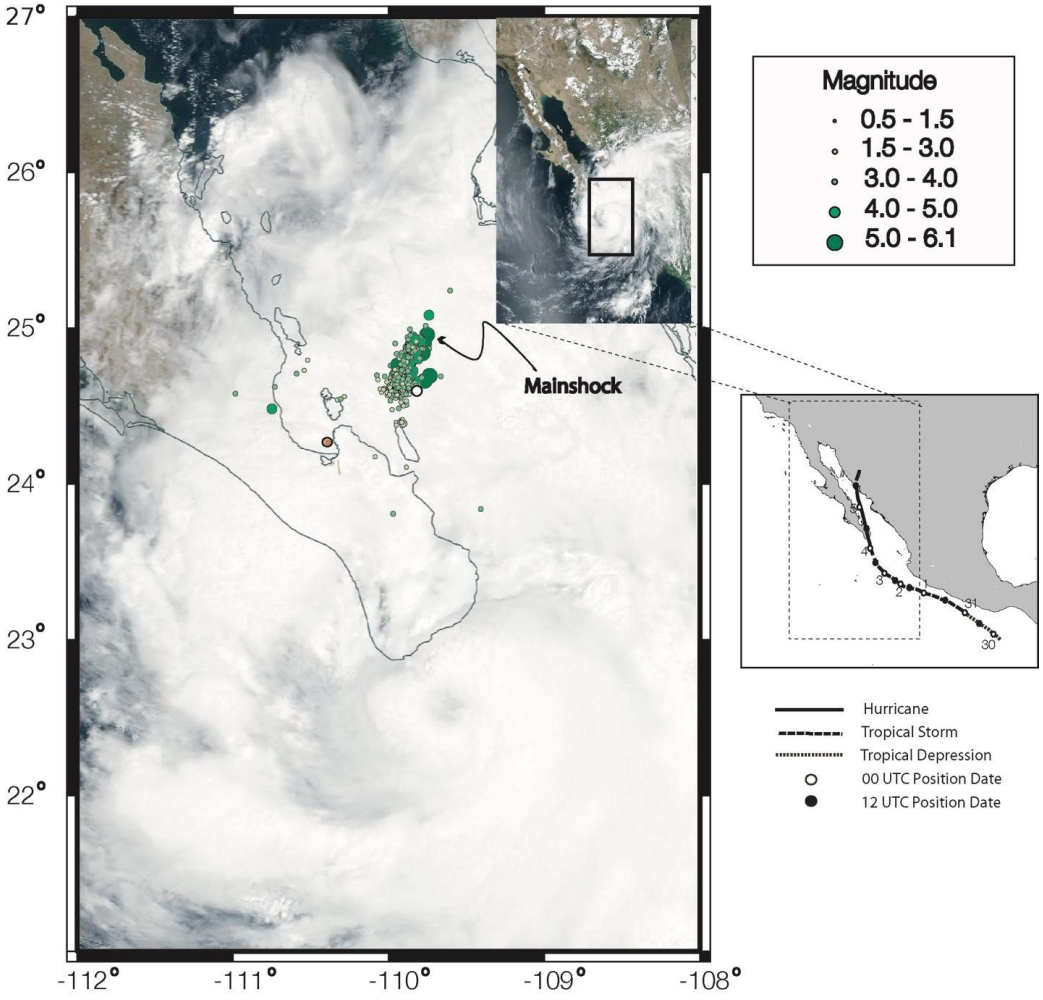
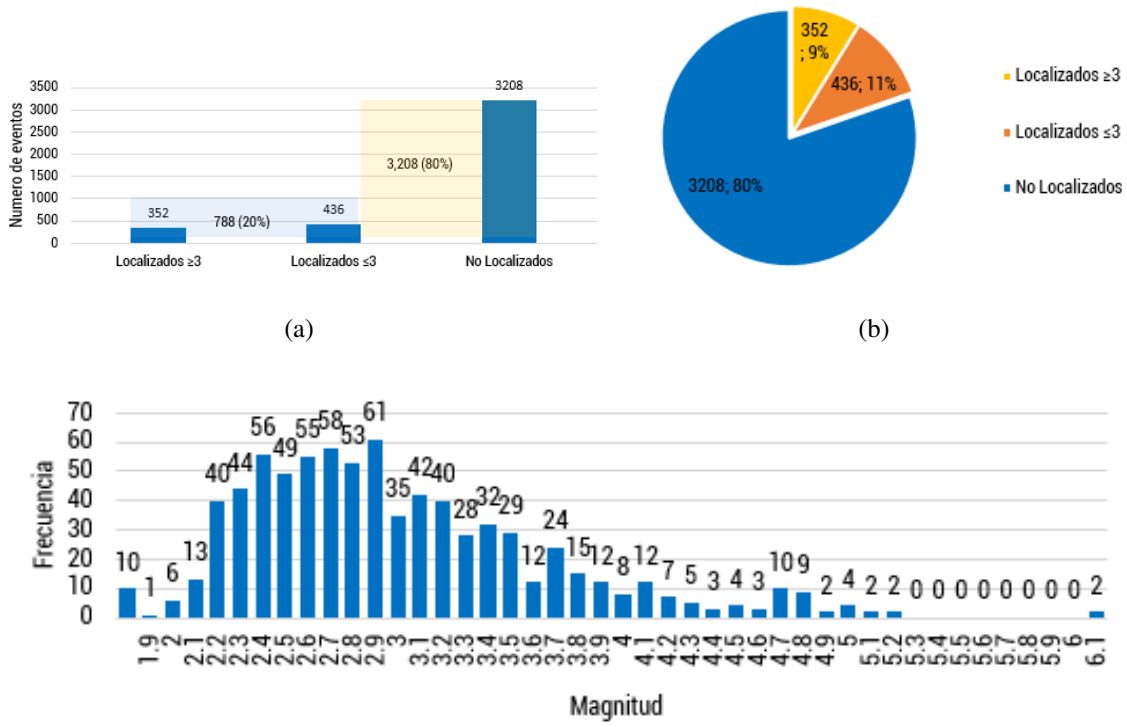


Figura 1.1.1: Imagen de satélite del Huracán Henriette y secuencia sísmica de los terremotos de Cerralvo Mw=6.1

1.3. Justificación

Como justificación se señala que los algoritmos de aprendizaje automático tienen:

- Uso relativamente nuevo que permiten identificar patrones sísmicos.
- Resultados en poco tiempo minimizando carga de trabajo para un analista humano.
- Representan una alternativa muy valiosa y un reto poder distinguir un terremoto del ruido sísmico.



(a) (b)

(c)

Figura 1.2.1: Distribución réplicas sismo de Cerralvo Mw= 6.1

1.4. Hipótesis

Es posible determinar cuando ocurre un sismo en registros continuos usando algoritmos de aprendizaje automático.

1.5. Objetivos

El objetivo del proyecto de investigación es clasificar las formas de onda sísmica registradas en sismogramas y determinar cuáles son sismos y cuales ruido utilizando un algoritmo de aprendizaje automático supervisado.

1.5.1. Objetivo general

Desarrollar un sistema automático para la clasificación de señales en registros sismicos.

1.5.2. Objetivo específico

Implementar algoritmos de aprendizaje automático que sean capaces de clasificar una señal sísmica en un registro.

1.6. Alcance y limitaciones

El estudio se realizará para las condiciones de la región de estudio, la parte sur del Golfo de California y con las estaciones de la red sísmica NARS Baja que estuvieron operando de agosto a diciembre de 2007 y como consecuencia de las réplicas del terremoto de Cerralvo de magnitud momento Mw = 6.1 ocurrido el día 1 de septiembre de 2007 y la presencia del huracán Henriette categoría 1.

Capítulo 2

MARCO TEÓRICO

2.1. Señal

Es una magnitud física mediante la cual se puede transmitir información, matemáticamente se puede representar como una función de una variable independiente t para el tiempo y que puede variar en amplitud y periodo [4].
Los parámetros básicos que caracterizan a una señal en el dominio del tiempo se muestran en la figura 2.1.1 y son:

- Amplitud: Es el valor que toma la señal en cada instante de tiempo.
- Periodo: es el tiempo transcurrido entre dos puntos equivalentes de la señal. Se dice que la señal es periódica si se repite cada cierto intervalo de tiempo.

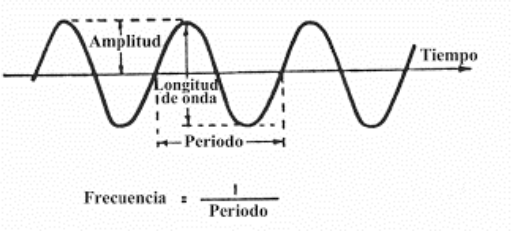


Figura 2.1.1: Características basicas de una señal.

2.1.1. Muestreo

El muestreo es el primer paso en el proceso de conversión de una señal analógica a una señal digital. La conversión de la señal análoga en digital se realiza, entre otras razones, porque las señales digitales presentan mayores ventajas al ser transmitidas y procesadas por computadoras como son mayor capacidad de reducir el ruido, mayor facilidad de procesamiento, etc. Las muestras se toman a intervalos de tiempo iguales, proceso denominado muestreo periódico de la señal [4]. El teorema de muestreo de Nyquist-Shannon establece que "la reconstrucción de una señal, del dominio temporal al dominio de frecuencia, es matemáticamente posible si, y solo si, la frecuencia de muestreo es mayor o igual al doble de la frecuencia límite máxima de la banda de la señal (ancho de banda)"

2.2. Señal sísmica

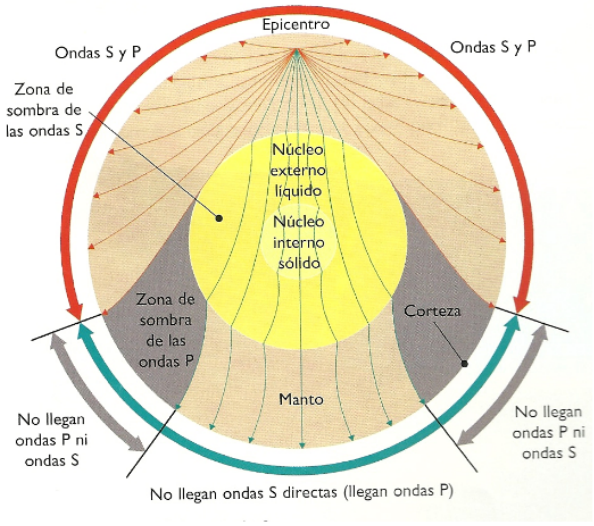
Una Señal sísmica es una forma de onda transitoria irradiada desde una fuente sísmica natural o artificial que puede utilizarse para localizar la fuente, analizar los procesos de la fuente y estudiar la estructura del medio de propagación [5].

Fases sísmicas

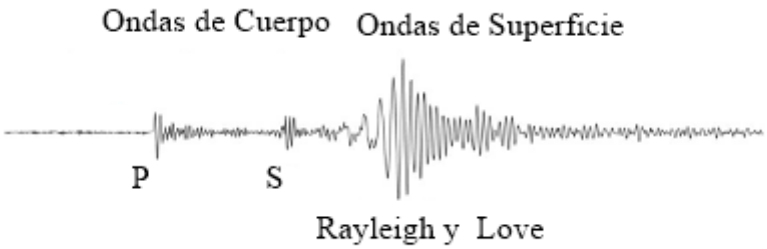
Los sismos generan ondas elásticas las cuales se desplazan a través de la tierra como se muestra en la figura 2.2.1.a . Existen 2 tipos de ondas elásticas: las ondas de cuerpo u ondas internas, que pueden ser compresionales o de cizalla, y las ondas superficiales, las cuales viajan por la superficie de la tierra y son más lentas, son causadas por las interferencias de las ondas de cuerpo que viajan en diferentes direcciones [6].
Cada tipo de onda en un sismograma se llama fase sísmica. Las principales se muestran en la figura 2.2.1.b.

Ondas P. Son ondas compresionales, se desplazan en la dirección de propagación de las ondas sísmicas como se muestra en la figura 2.2.2.a . Estas ondas son capaces de viajar a través de las rocas sólidas y de líquidos.

Ondas S. Estas ondas viajan más lento que las ondas P, son ondas de corte o de cizalla como se muestra en la figura 2.2.2.b. Se desplazan perpendicular a la dirección de propagación. Pueden viajar



(a) Propagación de las ondas de cuerpo y superficiales



(b) Fases sísmicas

Figura 2.2.1: Tipos de ondas sísmicas

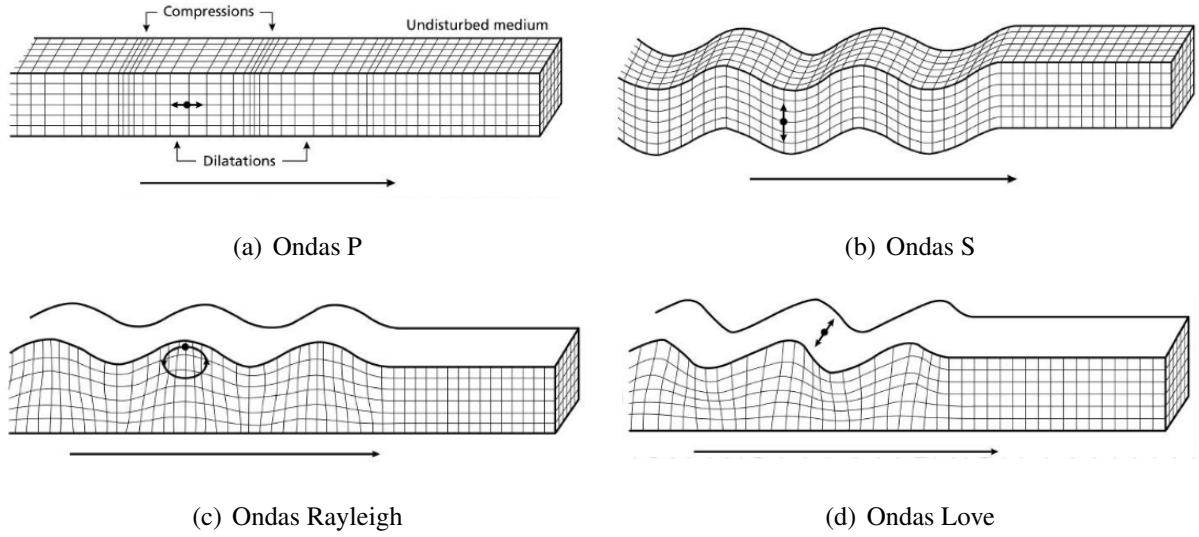
únicamente a través de sólidos debido a que los líquidos no pueden soportar esfuerzos de corte.

Ondas Rayleigh. Se deben a la interacción entre ondas P y S y al movimiento de cada partícula. Se da en forma de elipse retrógrada (figura 2.2.2.c)

Ondas Love. Es similar a ondas Rayleigh. Resultan de las interferencias constructivas polarizadas horizontalmente (figura 2.2.2.d)

La velocidad de propagación de la onda P es mayor que la de la onda S, por lo tanto, la primera vibración registrada es la onda P, y la segunda es la onda S.
La dirección de vibración de la onda P es la misma que la dirección de propagación, lo que significa que la onda P es más clara en el canal vertical.

La dirección de vibración de la onda S es perpendicular a la dirección de propagación, por lo que es más obvia en los canales horizontales.
Normalmente, la amplitud y el período de la onda P es menor que la de la onda S. Esto depende del patrón de radiación sísmico y del tipo de dislocación, en algunos casos como explosiones puras, donde no existe onda S, o bien son muy pequeñas.



(a) Ondas P

(b) Ondas S

(c) Ondas Rayleigh

(d) Ondas Love

Figura 2.2.2: Fases sísmicas

Espectro de ondas sísmicas

El rango de espectro y una clasificación de ondas sísmicas de banda ancha se muestra en la figura 2.2.3 Se muestra el rango de frecuencias para eventos locales (de 1 a 100 Hz) y para eventos telesísmicos (de 0.001 a 1 Hz) [5].

2.2.1. Ruido sísmico

El ruido sísmico es un término general usado para cualquier tipo de señal no deseada en los datos sísmicos. El ruido sísmico depende de los datos disponibles, del objetivo de estudio y del método de análisis. La razón principal detrás de los ruidos es tratar de evitar las perturbaciones indeseables de la señal mientras se acumulan datos en los sismómetros, sin embargo, en los últimos años, el ruido sísmico es una fuente importante de información en la exploración sísmica [7].
El ruido sísmico puede comprender:

- Vibraciones ambientales debidas a fuentes naturales como tormentas, viento, etc.

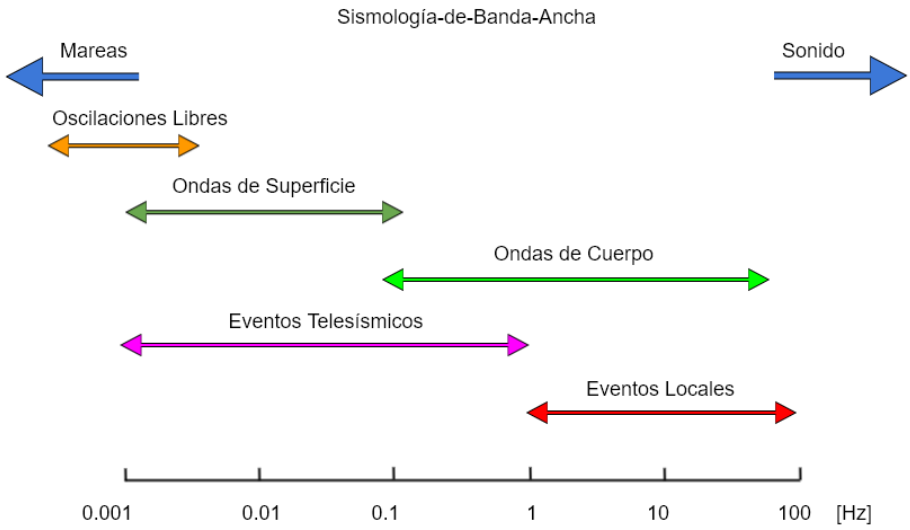


Figura 2.2.3: Rango de espectro de ondas sísmicas.

- Vibraciones provocadas por el hombre, en la industria, el tráfico, etc.
- Efectos de la gravedad como las aceleraciones horizontales debido a la inclinación de la superficie de la tierra.
- Señales debidas a la sensibilidad y a condiciones ambientales, como la temperatura, presión del aire, campo magnético, etc.
- Señales debidas a la calidad y deterioro por el uso del sensor sísmico, como la corrosión, deterioro de algunos componentes semiconductores, corrientes de fuga, etc.
- Ruido intrínseco del sismómetro, como el ruido electrónico y de cuantificación

Tipos de ruido sísmico

Los principales tipos de ruidos sísmicos se pueden dividir en 3 categorías:

1. Ruido por errores de instrumentación.

   Pueden deberse a errores de un instrumento defectuoso o vibraciones constantes cerca de los sismómetros que se registran debido a la sensibilidad. Dependen en gran medida del entorno y de los instrumentos donde se está grabando.

2. Ruido por actividades humanas y animales.

   Es el ruido que generalmente se debe a la interacción humana no natural con la tierra, como el tráfico de vehículos,actividades industriales, perforación de terrenos y construcción de edificios.

3. Ruido por fenómenos naturales.

   Es la interacción de las ondas sísmicas con las ondas de señal de fenómenos naturales como el efecto de marea del océano, flujo de un río, la corriente de viento o actividades volcánicas.

Eliminación de ruido

Para cada tipo de ruido, se requiere un nivel de estudio de la fuente que lo provoca y ajuste de los parámetros según los datos y el nivel de ruido. En general, la eliminación de la característica o componente no deseada de cualquier señal se conoce como filtro. En general existen 3 tipos de filtros de señal disponibles para los datos de un terremoto según la parte del espectro sobre la cual actúan:

1. **Filtro pasa alto.** Este filtro deja pasar todas las frecuencias altas definidas dentro de los parámetros y se eliminan las señales de baja frecuencia. Esto es muy útil en el caso de la señal que incluye el ruido constante de muy baja frecuencia, causado por el flujo de aire, corriente de agua o vibraciones constantes cerca de los sensores sísmicos.

2. **Filtro pasa bajo.** Este filtro deja pasar todas las frecuencias bajas y se restringen las frecuencias altas. Esto es útil en el caso de altas frecuencias abruptas registradas por los sensores sísmicos causados típicamente por algunas explosiones controladas, minería, volcanes, etc.

3. **Filtro pasa banda.** Es una combinación de filtro de pasa bajo y pasa alto, que es útil cuando hay ruidos de frecuencias altas y bajas en la señal.

2.3. Sismógrafo

Un sismógrafo es el instrumento utilizado para registrar las ondas generadas por los sismos. En ocasiones suele nombrarse de manera indistinta sismógrafo o sismómetro, sin embargo sismógrafo se refiere al conjunto de dispositivos y mecanismos que detectan, sincronizan y graban el movimiento de las ondas sísmicas y un sismómetro se refiere solo al sensor que detecta el movimiento.

El principio básico se puede explicar con un sistema mecánico, se basa en el principio de inercia que establece que todos los cuerpos tienen cierta resistencia al movimiento o a variar su velocidad, así,

el movimiento del suelo puede ser medido con respecto a la posición de una masa suspendida por un elemento que le permita permanecer en reposo por algunos instantes con respecto al suelo.

Este mecanismo se presenta en la figura 2.3.1, consiste de una masa suspendida de un resorte atado a un soporte acoplado al suelo, cuando el soporte se sacude al paso de ondas sísmicas, la inercia de la masa hace que ésta permanezca en reposo un instante y posteriormente cuando la masa sale del reposo, tiende a oscilar. Como esta oscilación del péndulo no refleja el verdadero movimiento del suelo, es necesario amortiguarla. Este amortiguamiento se logra por medio de una lámina sumergida en un líquido, generalmente aceite, posteriormente se hizo por medio de bobinas o imanes que ejercían las fuerzas amortiguadoras de la oscilación libre de la masa.

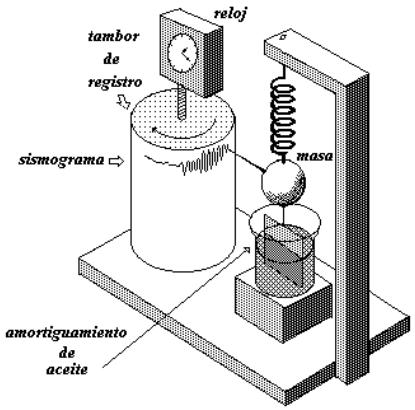


Figura 2.3.1: Principio de funcionamiento de un sismógrafo mecánico.

2.3.1. Sismógrafo de banda ancha

Son llamados así por la capacidad de registro de un mayor rango de frecuencias. La mayoría de estos sensores sísmicos utilizan un sistema de fuerza de retroalimentación o sistema de fuerza balanceada. Este consiste de un circuito de retroalimentación negativo, el cual ejerce una fuerza proporcional al desplazamiento de la masa inercial a fin de cancelar el movimiento relativo y por medio de un transductor eléctrico convierte el movimiento de la masa en una señal eléctrica, la cual es una estimación de la fuerza de retroalimentación que debe ejercerse para anular el movimiento.

Estos sistemas permiten extender el ancho de banda y la linealidad de los sismómetros porque no

permiten grandes movimientos de la masa que doblen los resortes o los niveles. La señal de salida de estos sistemas posee gran rango dinámico debido a que los transductores electromagnéticos tienen un amplio rango dinámico [5].

Como el movimiento del suelo tiene lugar en las tres dimensiones, se requieren arreglos de 3 sensores ortogonales, dos componentes horizontales y una componente vertical. El Sensor STS-2 que se muestra en la figura 2.3.2 es el utilizado en las estaciones IRIS / IDA GSN Incorporated Research Institutions for Seismology/ International Deployment of Accelerometers Global Seismographic Network es un sismómetro triaxial de banda ancha fabricado por G. Streckeisen AG. Diseñado para una rápida y fácil instalación, amplio rango de temperatura de operación, de tamaño compacto y seguro de transportar.

El esquema simplificado del sismómetro de banda ancha STS-2 se muestra en la figura 2.3.3. Se puede observar como el desplazamiento de masa (M) en relación con la carcasa del instrumento se detecta mediante un transductor de desplazamiento capacitivo (K) y se convierte en una señal eléctrica que se transmite a la bobina de retroalimentación (L).

Figura 2.3.2: Sensor STS-2 utilizado en las estaciones IRIS / IDA GSN. Es un sensor triaxial de banda ancha fabricado por G. Streckeisen AG.

Estación Sísmica

Una estación sísmica es una instalación que se ubica de ser posible en roca firme generalmente en un sitio aislado a fin de minimizar las condiciones de ruido ambiental como el viento, cursos de agua, lejos de núcleos urbanos y de fuentes de ruido cultural como el tráfico, maquinaria, actividades agrícolas e industriales, tendidos eléctricos, etc.

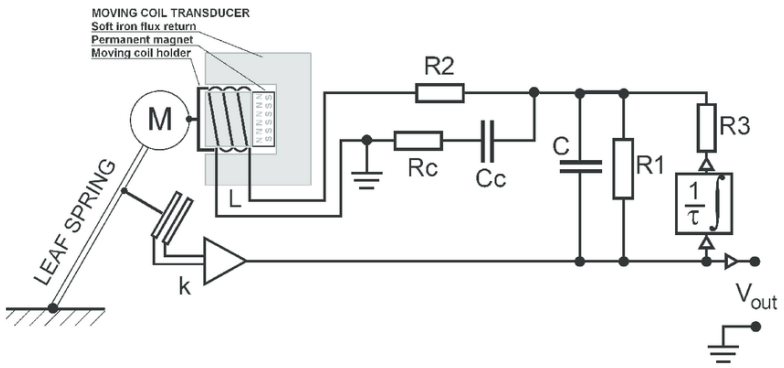


Figura 2.3.3: Esquema simplificado del sismómetro de banda ancha STS-2.

Consta principalmente de una caseta y un sensor aislados térmicamente, un equipo de registro que permite la adquisición de la señales y su almacenamiento (Datalogger), puede incluir la transmisión en tiempo real hacia un centro de adquisición de datos. Debido al bajo consumo de energía cuentan con sistemas de alimentación por medio de sistemas fotovoltaicos y son de operación autónoma. La figura 2.3.4 muestra una estación sísmica típica.

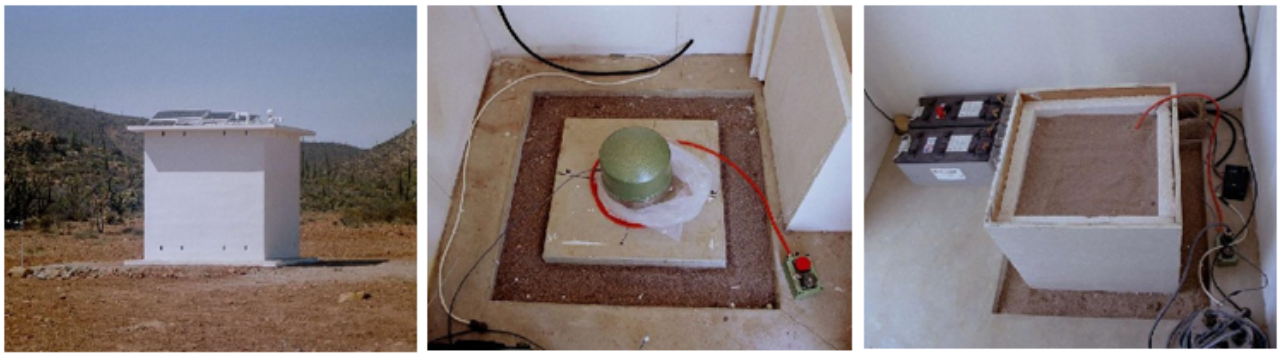


Figura 2.3.4: Estación sísmica típica de la Red NARS-Baja.

Red Sísmica NARS – Baja.

La red Sísmica NARS- *Baja Network of Autonomously Recording Seismographs* fue la red sísmica del grupo de sismología de la Facultad de Geociencias de la Universidad de Utrecht, Holanda operada por CICESE de 2002 a 2008. Como se muestra en la figura 2.3.5, esta red consistió de 19 estaciones sísmicas de banda ancha a ambos lados del Golfo de California formando una matriz de más de 4000 km de extensión con distancia promedio entre estaciones de 100-150 km. Su objetivo era comprender el proceso de separación de las grietas del Golfo de California que unen el sistema de fallas de San

Andreas en California con el sistema de expansión oceánica del Pacífico Oriental [8].

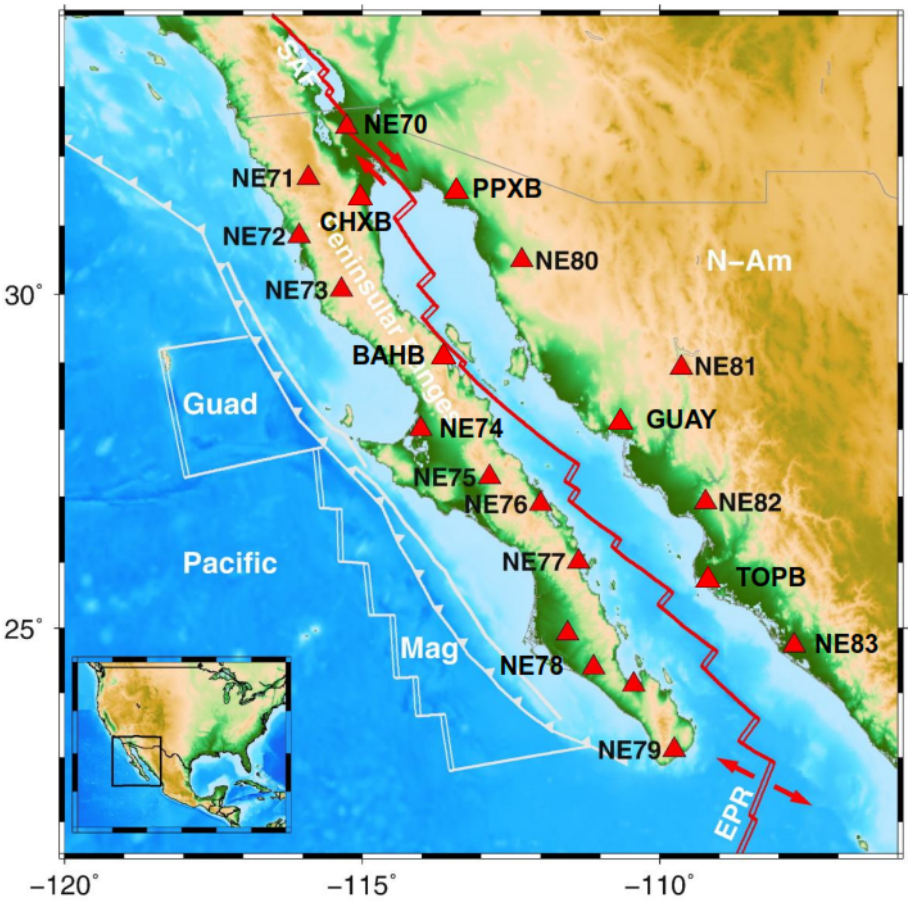


Figura 2.3.5: Mapa de la región del Golfo de California, principales fallas geológicas y estaciones de la Red Sísmica NARS-Baja.

Las estaciones sísmicas de la Red NARS-Baja constaban de:

- Sensor. De banda ancha Streckeisen STS-2. Registra 3 componentes a 20 sps (*samples per second*).
- Timing. Antena GPS marca Trimble Acutime 2000. Sincronizada con UTC (*Coordinated Universal Time*).
- Digitalizador (*Datalogger*)

Se muestran a continuación en la figura 2.3.6.

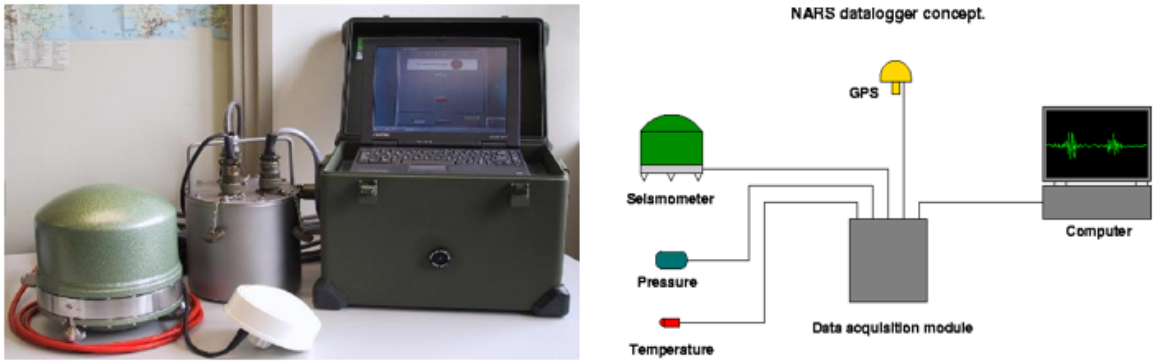


Figura 2.3.6: Sensor sísmico y digitalizador de una estación típica de NARS-Baja.

2.4. Procesamiento digital de una señal sísmica

Consiste en la transformación de una señal sísmica a un conjunto de vectores de características. Se divide en dos etapas: preprocesamiento de la señal y extracción de características.

2.4.1. Preprocesamiento de la Señal

Incluye los procesos de adquisición, corrección instrumental, filtrado y normalización, lo que permitirá preparar las señales sísmicas para la fase de extracción de características en el dominio del tiempo y de la frecuencia.

2.4.2. Extracción de características

Son técnicas que transforman las señales a vectores de características, seleccionando un conjunto de atributos tanto en el dominio del tiempo como de la frecuencia evitando datos redundantes o que no aportan información, con lo que se logra una reducción de dimensionalidad [9].

2.5. Aprendizaje Automático

Es un conjunto de técnicas que forman parte de la inteligencia artificial que proporciona a las computadoras la habilidad de aprender y mejorar automáticamente a partir de la experiencia sin estar programados explícitamente. Se centra en el desarrollo de programas que pueden acceder a datos para utilizarlos y aprender por sí mismos. A través del aprendizaje automático las computadoras son capaces de detectar patrones en un conjunto de datos y realizar predicciones [10].

El objetivo del aprendizaje automático es reducir la diferencia entre un resultado de una predicción y un resultado real. A esto también se le denomina función de pérdida o función de costo. El objetivo es minimizar la función de perdida al encontrar el valor optimizado de un modelo asegurándose que un algoritmo generalice bien.

2.5.1. Tipos de aprendizaje automático

El aprendizaje automático se puede clasificar según el tipo y la cantidad de supervisión humana durante el entrenamiento en 2 categorías principalmente [11]:

- **Aprendizaje supervisado.**

  En los algoritmos de aprendizaje supervisado se genera un modelo basado en datos de entrada y salida previamente etiquetados y clasificados, en el que se sabe previamente a cual grupo, valor o categoría pertenecen las instancias, de estos se forma un conjunto de datos llamados datos de entrenamiento, con los que se realiza el ajuste de un modelo inicial, de tal manera que el algoritmo va ajustando los resultados de las muestras a clasificar comparando el resultado del modelo con la etiqueta real, ajustando y compensando el modelo hasta ajustar el resultado a un minimo de errores. Después de comprender los datos, el algoritmo determina qué etiqueta se debe asignar a los nuevos datos asociando patrones a los nuevos datos sin etiquetar.

  Los algoritmos más importantes de aprendizaje supervisado son:

  - Clasificador bayesiano ingenuo (Naive Bayes)
  - K-vecinos cercanos
  - Regresión logística
  - Redes neuronales
  - Máquinas de vector soporte
  - Regresión logística
  - Árboles de decisión y bosques aleatorios

- **Aprendizaje no supervisado.**

  Los algoritmos de aprendizaje no supervisado ajustan un modelo tomando en cuenta los datos de entrada, sin importar los de salida, es decir, a diferencia del supervisado utiliza datos que

no están clasificados ni etiquetados y el algoritmo actúa agrupando los datos según similitudes, patrones y diferencias. Dentro de este tipo de algoritmos, el agrupamiento o clustering es el más utilizado, ya que particiona los datos en grupos que poseen características similares entre sí.
Los algoritmos no supervisados más importantes son:

- Clustering: k-means, análisis de clúster jerárquico.
- Aprendizaje de reglas de asociación
- Visualización y reducción de dimensionalidad: kernel PCA, PCA.

A su vez, el aprendizaje supervisado se puede dividir en dos categorías: en algoritmos de clasificación y de regresión.

- **Algoritmos de Clasificación.**
  Los algoritmos de clasificación se usan cuando el resultado deseado es una etiqueta discreta, es decir, la respuesta al problema cae dentro de un conjunto finito de resultados posibles. Cuando el modelo es para predecir dos clases, verdadero o falso, se le conoce como clasificación binaria. Existen varias técnicas de aprendizaje automático en problemas de clasificación dentro de los que se pueden mencionar:

  - Regresión logística.
  - Máquinas de vector soporte.
  - Arboles de decisión.
  - Bosques aleatorios.
  - Redes neuronales y aprendizaje profundo.

- **Algoritmos de Regresión**
  La regresión es útil para predecir valores que son continuos, la salida se representa mediante una cantidad que se determina en función de las entradas en lugar de limitarse a un conjunto de etiquetas. Algunas de sus características son:

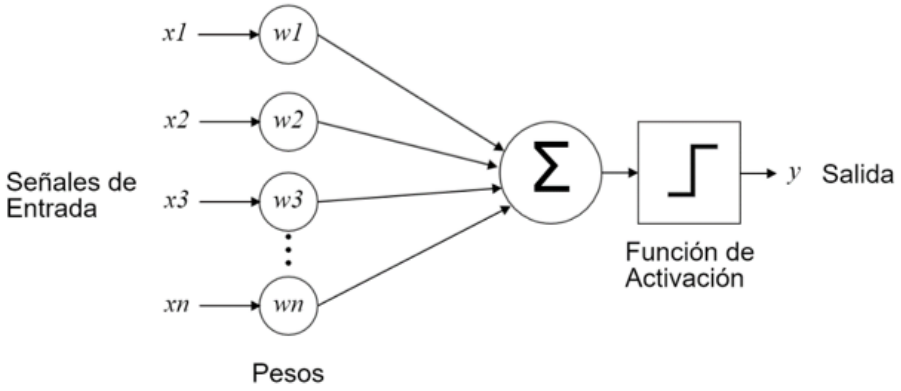
  - Establece un modelo mediante una gráfica de dispersión.
  - Señalan una tendencia en los datos

Existen varios algoritmos de aprendizaje automático en problemas de regresión dentro de las que se pueden destacar:

- Regresión lineal y regresión no lineal.
- Máquinas de vector soporte.
- Árboles de decisión.
- Bosques aleatorios
- Redes neuronales y aprendizaje profundo

2.5.2. Perceptrón

El perceptrón es la red neuronal más básica formada de una neurona artificial o nodo. Su funcionamiento es muy sencillo, recibe una o varias entradas, cada entrada tiene un peso asociado w que se va modificando en el proceso de aprendizaje, realiza la suma ponderada de todas las entradas de acuerdo a los pesos y el resultado lo introduce en una función llamada de activación que genera el resultado final.Se muestra en la figura 2.5.1.
El entrenamiento consiste en determinar los pesos w y el umbral de activación que mejor ajuste los valores deseados y los calculados por la red. El resultado puede servir como entrada de otras unidades [12].



$$y_i = \sum_j w_{ij}\, y_i$$

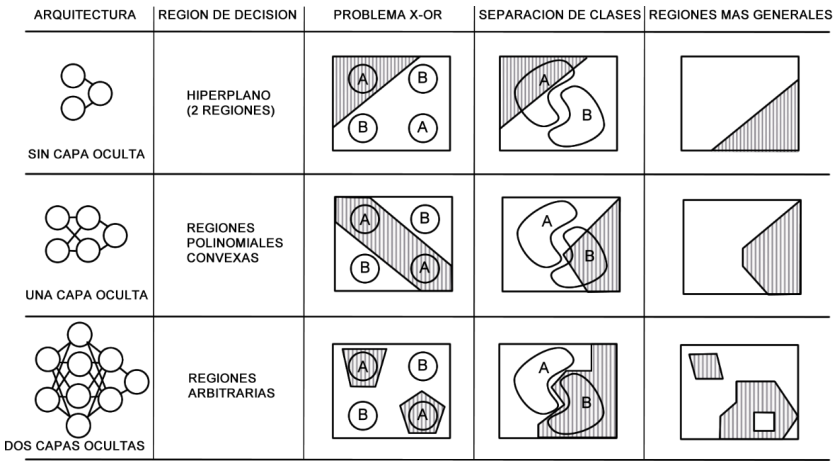
Figura 2.5.1: Esquema básico de un perceptrón.

2.5.3. Perceptrón multicapa

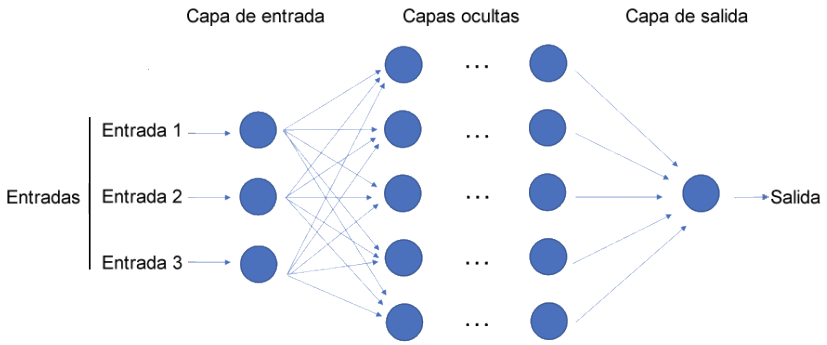
Un perceptrón multicapa contiene nodos organizados en capas para formar una Red Neuronal Artificial ANN. En una ANN de una sola capa cada nodo se conecta directamente a una variable de entrada y contribuye a una variable de salida. Una red de una sola capa se puede extender a una red de múltiples capas denominado Perceptrón Multicapa MLP. Los tipos de capas en un MLP son:

- Capa de entrada: variables de entrada, llamada capa visible.
- Capas ocultas: capas de nodos entre las capas de entrada y salida. Puede haber una o más capas.
- Capa de salida: una capa de nodos que producen las variables de salida.

El perceptrón multicapa puede usarse para representar regiones convexas, es decir, que puede ajustarse a formas alrededor de un espacio de alta dimensionalidad que pueda separar y clasificar diferentes clases, superando la limitación del perceptrón de la separabilidad lineal [13]. Un MLP con 2 capas ocultas como se muestra en la figura 2.5.2. es suficiente para crear regiones de clasificación de cualquier forma deseada, de acuerdo al hallazgo teórico mencionado en el artículo “An introduction to computing with neural nets” [14].



(a) Tipos de regiones de decisión de perceptrones de varias capas



(b) Perceptrón multicapa MLP

Figura 2.5.2: Regiones de decisión de acuerdo al numero de capas de un MLP

2.5.4. Red Neuronal Artificial

Una red neuronal artificial es un modelo computacional simplificado inspirado en las redes neuronales biológicas que buscan emular su comportamiento. Consiste de un conjunto de neuronas artificiales interconectadas y a partir de señales de entrada generan una salida. Las unidades de procesamiento se organizan en capas, una capa de entrada, que representan las entradas, una o varias capas ocultas, y una capa de salida [15]. En esta tesis se nombrarán redes neuronales, a las redes neuronales artificiales. Las principales características de las redes neuronales son:

- Autoorganización y Adaptabilidad. Utilizan algoritmos de aprendizaje adaptativo y de autoorganización, por lo que ofrecen mejores posibilidades de procesado robusto y adaptativo
- Procesado no lineal. Aumenta la capacidad de la red para aproximar funciones, clasificar patrones y aumentar su inmunidad frente al ruido.

- Procesado paralelo. Normalmente se usa un gran número de nodos de procesado, con alto nivel de interconectividad.

Una red neuronal artificial es una MLP que tiene una capa de entrada que se conecta a las variables de entrada, una o más capas ocultas y una capa de salida que produce las variables de salida.
Una red neuronal de una sola capa solo se puede utilizar para representar funciones separables linealmente, lo que significa problemas muy simples en los que 2 clases en un problema de clasificación pueden estar claramente separadas por una línea. Sin embargo la mayoría de los problemas en el mundo real no son linealmente separables.

En general se puede describir la forma y la capacidad de una red neuronal, de acuerdo a su:

- Tamaño: Número de nodos en un modelo.

- Ancho: Número de nodos en una capa. Una forma de incrementar la capacidad del modelo es crear una red más ancha.

- Profundidad: Número de capas en una red neuronal. Una manera de mejorar el desempeño de una red neuronal es crear una red neuronal más profunda al añadirle capas, lo cual permitirá al modelo extraer más cantidad de características del conjunto de datos de entrada.

  Una red neuronal profunda es simplemente una red neuronal artificial con varias capas ocultas entre las capas de entrada y de salida. Las capas adicionales proporcionan un gran aumento en la potencia computacional, lo que permite que las redes neuronales profundas alcancen un rendimiento sorprendente en múltiples tareas, y puedan modelar relaciones no lineales complejas.

  Desde un punto de vista práctico, el aprendizaje profundo es muy bueno para extraer características, por ejemplo el sistema visual humano contiene múltiples extractores de características, contamos con sistemas dedicados para detectar bordes, formas simples y características de mayor nivel.

  Crear un sistema que haga lo mismo es muy difícil, aquí es donde entran las redes neuronales profundas [16]. Estas han podido resolver problemas como la visión y audio por computadora y el procesamiento avanzado de lenguaje natural, pero a cambio requieren de grandes conjuntos de datos, de muchos ajustes y de mayor potencia informática.

- Capacidad: Tipo o estructura de funciones que puede aprender una configuración de red llamada en ocasiones capacidad de representación.

- Arquitectura: Disposición específica de las capas y nodos en la red.

2.5.4.1. Parámetros e Hiperparámetros

Dos términos en el aprendizaje automático a menudo que se confunden son parámetros de modelo e hiperparámetros:

- **Parámetro.** Un parámetro es una variable interna de un modelo de aprendizaje automático, no se establece manualmente, y su valor se puede estimar o extraer de los datos durante el entrenamiento de los conjuntos de datos históricos, se guardan como parte del modelo aprendido y son requeridos al hacer predicciones.

- **Hiperparámetro.** Los hiperparámetros son parámetros ajustables que permiten controlar el proceso de entrenamiento de un modelo, su valor no se puede estimar a partir de los datos, si no que se establece antes de que el modelo comience a entrenarse. El valor óptimo no se puede conocer a priori para un problema dado, pero se pueden usar reglas generales, los valores que han funcionado anteriormente en problemas similares o seleccionar un valor a prueba y error.
  Los hiperparámetros afectan la velocidad y precisión del proceso de aprendizaje. Su ajuste consiste en encontrar la configuración de hiperparámetros que produzca el mejor rendimiento, normalmente este proceso es manual y costoso desde el punto de vista computacional [17].
  Algunos ejemplos de hiperparámetros utilizados para entrenar los modelos son:

  - Tasa de aprendizaje
  - Número de iteraciones
  - Número de vecinos en k-vecinos más cercanos
  - Profundidad máxima en un árbol de decisión
  - Hiperparámetros C y sigma en máquinas de vector soporte
  - Número de capas en una red neuronal
  - Número de neuronas por capa en una red neuronal
  - Entre otros

Las diferencias mas importantes entre estos 2 terminos se presentan en la tabla 2.5.1

Parámetro	Hiperparámetro
Estimado durante el entrenamiento con datos históricos.	Los valores se establecen de antemano.
Es parte del modelo.	Externo al modelo.
Se estiman mediante algoritmos de optimización.	Se estiman mediante el ajuste de hiperparámetros.
El valor estimado se guarda con el modelo entrenado.	No forma parte del modelo entrenado y, por lo tanto, los valores no se guardan.
Depende del conjunto de datos con el que se entrena el sistema.	Independiente del conjunto de datos.
Son necesarios para hacer predicciones.	Son necesarios para estimar los parámetros del modelo.

Tabla 2.5.1: Diferencias entre parámetros e hiperparámetros

2.5.4.2. Número de capas y nodos en una red neuronal

El número de capas y de nodos en cada capa son los hiperparámetros de un modelo de Red Neuronal Artificial que es necesario especificar. El mayor problema al que se enfrenta al diseñar un modelo ANN es decidir cuántas capas debe usar el perceptrón multicapa y cuántos nodos por capa. En general existen 5 enfoques que dan un punto de partida [18], y tratan de resolver este problema:

1. **Experimentación.** No se puede calcular analíticamente la cantidad de capas o la cantidad de nodos que se usarán por capa en una red neuronal artificial para abordar un problema específico de modelado predictivo del mundo real. Por lo que se deben de hacer pruebas con varios experimentos controlados. Independientemente de la heurística todas las respuestas volverán a la necesidad de una experimentación para decidir con cuantas capas y neuronas funciona mejor para un conjunto de datos específico.

2. **Intuición.** Un modelo profundo proporciona una jerarquía de capas que acumulan mayores niveles de abstracción desde el espacio de las variables de entrada a las variables de salida. Una vez comprendido un problema, podemos intuir que se requiere un modelo profundo para resolver un problema de predicción, por lo que se puede elegir una configuración de red que tenga muchas capas de profundidad. Esta decisión de tomar una decisión puede provenir de la experiencia con el dominio de este tipo de problemas, la experiencia de modelado con redes neuronales, o una combinación de ambas.

3. **Ir a la profundidad.** Empíricamente las redes neuronales profundas parecen funcionar mejor.

Una mayor profundidad puede resultar en una mejor generalización para una amplia variedad de tareas. Por lo que siguiendo este enfoque heurístico es sugerible el uso de redes profundas para resolver problemas de modelado predictivo más complejos.

4. **Pedir ideas prestadas.** Otro enfoque es aprovechar los hallazgos reportados en la literatura. Consiste en buscar trabajos de investigación que describan el uso de MLPs en casos de problemas de predicción similares tomando en cuenta la configuración de las redes utilizadas en esos documentos y utilizarlas como punto de partida para las configuraciones de prueba del problema.

5. **Buscar.** Diseñar un experimento de búsqueda automatizada para probar diferentes configuraciones de red. Algunas estrategias son:

    - Aleatoria: prueba configuraciones aleatorias de capas y nodos por capa.
    - Cuadrícula: búsqueda sistemática en el número de capas y nodos por capa.
    - Heurística: intentar una búsqueda dirigida a través de configuraciones como un algoritmo genético u optimización bayesiana.
    - Exhaustiva: factible para redes y conjuntos de datos pequeños. Consiste en realizar el máximo número de combinaciones de capas y de nodos, hasta donde sea posible.

2.5.4.3. Redes Neuronales Artificiales utilizando Keras

Keras es una biblioteca de Redes Neuronales de código abierto y de alto nivel escrita en Python y capaz de ejecutarse sobre TensorFlow [19].

- Permite la creación de prototipos fácil y rápidamente a través de la facilidad de uso, modularidad y extensibilidad.
- Admite redes convolucionales y redes recurrentes, así como combinaciones de las dos.
- Se ejecuta sin problemas en CPU y GPU.
- Es compatible con: Python 2.7-3.+

Se rige bajo los siguientes principios:

- Facilidad de uso. Keras es una API diseñada para seres humanos, no para máquinas.
- Modularidad. Módulos independientes totalmente configurables que se pueden conectar: capas, funciones de costo, optimizadores, inicialización, funciones de activación, etc. Estos módulos son independientes y pueden combinarse para crear nuevos modelos.
- Fácil extensibilidad. Se puede agregar fácilmente nuevos módulos a los ya existentes
- Trabaja con Python. Los modelos se desarrollan en Python, son compactos, fáciles de depurar y permiten la facilidad de extensibilidad.

2.5.4.3.1. Función de activación

La función de activación decide si una neurona debe activarse o no mediante el cálculo de la suma ponderada y añadiendo un sesgo adicional. El propósito de la función de activación es introducir la no linealidad en la salida de una neurona [20].
Existen varias funciones de activación, las más utilizadas para problemas de clasificacion son sigmoide y softmax. Aunque ambas funciones son las mismas a nivel funcional, en general se usa softmax cuando hay un número mayor a 2 clases, aunque ambas pueden usarse para la clasificación binaria.
Las principales diferencias entre ambas se muestran en la tabla 2.5.2.

Función Sigmoide. La función Sigmoide se utiliza mayormente para la clasificación binaria y en modelos de regresión logística. Tiene su dominio en los números reales, y devuelve un valor en el rango de 0 a 1 (figura 2.5.3). Otro rango de uso común es de -1 a 1.

Función Softmax

La función Softmax calcula la probabilidad de cada clase objetivo sobre otras clases posibles, las probabilidades calculadas determinaran la clase objetivo para las entradas dadas. La principal ventaja de usar Softmax es el rango de probabilidades de salida. El rango será de 0 a 1, y la suma de todas las probabilidades será igual a uno.
Soporta sistemas de clasificación multinomial, por lo que se convierte en el recurso principal utilizado en las capas de salida de un clasificador. Es utilizada en modelos de regresión logística de clasificación múltiple.

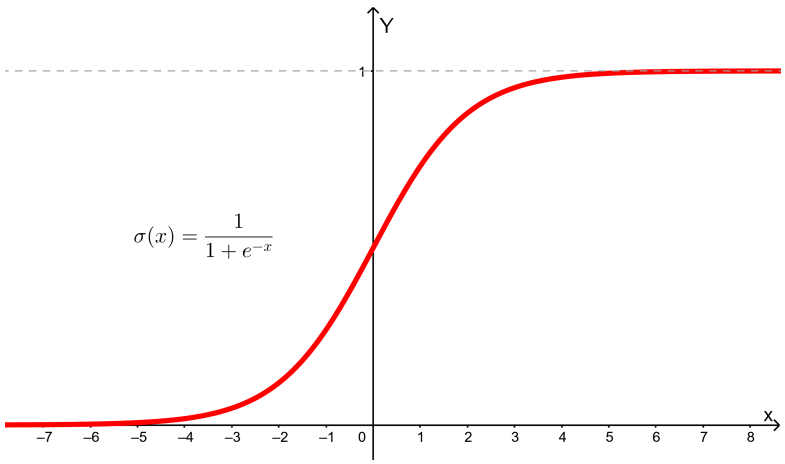


Figura 2.5.3: Función sigmoide.

Diferencias entre la función Sigmoide y la función Softmax

Función Sigmoide	**Función Softmax**
Utilizada para clasificación binaria en modelos de regresión logística.	Utilizada para clasificación múltiple en modelos de regresión logística.
La suma de probabilidades no necesita ser 1.	La suma de probabilidades debe ser 1.
Se utiliza como función de activación al construir redes neuronales.	Utilizada en las diferentes capas de redes neuronales.
El valor más alto tendrá la alta probabilidad pero no la mayor probabilidad.	El valor más alto tendrá la mayor probabilidad sobre otros valores.

Tabla 2.5.2: Diferencia entre la función Sigmoide y la función Softmax.

2.5.4.3.2. Función de Perdida

La función de perdida (o función de costo) es una medida de cómo se cuantifican las diferencias entre las variables observadas y predichas. La función de pérdida debe minimizarse para obtener los mejores valores para cada parámetro del modelo [21]. Keras permite tres métricas simples para tratar variables de resultado cuantitativas, binarias o multiclase :

1. Funciones de pérdida por regresión (*Regression Loss*).
2. Funciones de pérdida de clasificación binaria (*Binary Classification Loss*).
3. Funciones de pérdida de clasificación de clase múltiple (*Multi-class Classification Loss*).

Funciones de pérdida para clasificación binaria

Se utilizarán funciones de pérdida de clasificación binaria, ya que el problema es de clasificacion de 2 clases, sismo o ruido. Las funciones de pérdida más comunes que se pueden usar al entrenar modelos de red neuronal para la clasificación binaria son:

Entropía cruzada (*binary_crossentropy*). También conocida como pérdida logarítmica, esta función calcula la entropía cruzada en problemas de clasificación binaria. Es la función de pérdida predeterminada que se usa para problemas de clasificación binaria.
La clasificación binaria son aquellos problemas de modelado predictivo en los que a los ejemplos se les asigna una de dos etiquetas. El problema es asignar a una predicción un valor de 0 o 1 para la primera o segunda clase.

Pérdida *Hinge Loss*. Una alternativa a la entropía cruzada para problemas de clasificación binaria es la función *Hinge Loss*. Se usa principalmente para la clasificación binaria donde los valores objetivos están en el conjunto -1, 1. Esta función de pérdida alienta a los ejemplos a tener el signo correcto, asignando más error cuando hay una diferencia en el signo entre los valores de clase real y predicha.
Los informes de rendimiento son mixtos, lo que a veces resulta en un mejor rendimiento que la entropía cruzada en problemas de clasificación binaria.

Efectos de la función de pérdida con diferentes tasas de aprendizaje.

La tasa de aprendizaje determina la velocidad con la que se aproxima a un valor minimo de error. Mientras mas rápido se llegue al mínimo más rápido estará aprendiendo la Red Neuronal a realizar su tarea. Sin embargo valores muy altos de tasas de aprendizaje pueden tener el efecto contrario, llevando a que el algoritmo se pase de largo su valor mínimo, lo que impide el aprendizaje, por lo que en ocasiones se requiere ir mas lento con bajas tasas de aprendizaje.
Como se muestra en la figura 2.5.4, con bajas tasas de aprendizaje, las mejoras tienen una tendencia lineal (línea azul), con altas tasas la tendencia será exponencial (linea roja), pero con las tasas de aprendizaje más altas disminuirán más rápido, pero se estancarán en los peores valores de pérdida (línea verde).

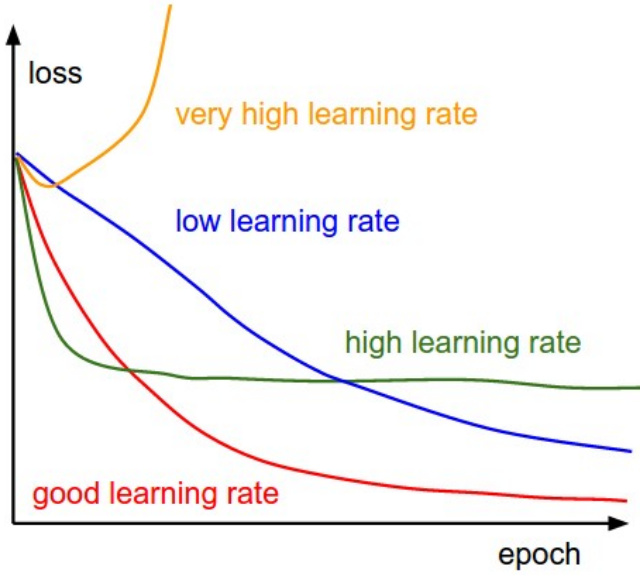


Figura 2.5.4: Efectos de diferentes tasas de aprendizaje.

2.5.4.3.3. Función de Optimización Los optimizadores, en combinación con la función de pérdida, son las piezas clave utilizadas para cambiar algunos atributos en una red neuronal, como los pesos y la tasa de aprendizaje. Los algoritmos de optimización se encargan de reducir las pérdidas y proporcionar los resultados más precisos posibles.

Existen numerosos optimizadores y no existe una regla clara sobre cuál es el mejor. Los principales optimizadores disponibles son:

a. **Descenso de gradiente estocástico (SGD).** Es uno de los algoritmos numéricos más populares para optimizar una pérdida. La forma más simple de actualización es cambiar los parámetros a lo largo de la dirección negativa del gradiente (ya que el gradiente indica la dirección del aumento, pero generalmente lo que se desea es minimizar una función de pérdida). Funciona bien para redes poco profundas.

   Existen 3 variantes de Descenso de Gradiente:

   1. **Descenso de Gradiente por Lotes.** Calcula el gradiente de la función de pérdida para todo el conjunto de datos de entrenamiento.
   2. **Descenso de Gradiente Estocástico (SGD).** Particiona aleatoriamente el conjunto de datos en subconjuntos y actualiza el gradiente usando un subconjunto único, luego el siguiente lote se usa para la siguiente iteración. Existen algunas variantes de SGD:

- **Momentum:** Momentum ayuda a acelerar el descenso de gradiente cuando se tienen funciones con superficies que se curvan más abruptamente en una dirección que en otra. También amortigua la oscilación. Tiene en cuenta los gradientes pasados para suavizar los pasos del descenso del gradiente. Se puede aplicar con descenso de gradiente de lote, descenso de gradiente de mini lote o descenso de gradiente estocástico.
- **Nesterov:** Es un algoritmo predictor-corrector que generalmente supera al estimador Momentum. Se implementa en dos pasos: en la etapa de predicción, la trayectoria se extrapola linealmente como en el Momentum, en la segunda etapa, se corrige dando como resultado una aceleración de convergencia.

3. **Descenso de Gradiente Minibatch.** Es una combinación de los dos métodos y se basa en dividir el conjunto de datos de entrenamiento en pequeños lotes. El gradiente se promedia sobre un pequeño número de muestras que permiten reducir el ruido y la aceleración de la velocidad.

b. **Algoritmos de optimización con tasas de aprendizaje adaptativo**

- **Adagrad.** Adagrad es un método de tasa de aprendizaje adaptativo. Se basa en gradiente que adapta la relación de aprendizaje a los parámetros: Realiza grandes actualizaciones para parámetros poco frecuentes y pequeñas actualizaciones para parámetros frecuentes. Por esta razón, es muy adecuado para tratar con datos escasos.
  La principal ventaja de Adagrad es que no se requiere ajustar la tasa de aprendizaje manualmente. Su principal desventaja es que su tasa de aprendizaje siempre está disminuyendo y decayendo.
- **AdaDelta.** Es una extensión de AdaGrad que busca reducir la tasa de aprendizaje agresiva y monotónicamente decreciente que genera AdaGrad.
  La ventaja de AdaDelta es que no se requiere establecer una tasa de aprendizaje predeterminada
- **RMSprop.** RMSprop Propagación de Raíz Media Cuadrática es un algoritmo de tasa de aprendizaje adaptativo que combina SGD y propagación de raíz media cuadrática. RMSprop intenta resolver las tasas de aprendizaje decrecientes de Adagrad utilizando un promedio móvil del gradiente cuadrado. RMSProp divide la tasa de aprendizaje por el promedio de la disminución exponencial de gradientes cuadrados, ajustándose automáticamente para

cada parámetro.

RMSprop y AdaDelta se han desarrollado de manera independiente al mismo tiempo debido a la necesidad de resolver las tasas de aprendizaje radicalmente decrecientes de Adagrad.

- **Adam.** Adam Estimación Adaptativa de Momentum, se puede ver como una combinación de RMSprop y descenso de gradiente estocástico con impulso RMSprop + SGDMomentum.

  Es otro método que calcula las tasas de aprendizaje adaptativo para cada parámetro. Esta diseñado específicamente para entrenar redes neuronales profundas, es muy eficiente computacionalmente hablando y requiere poca capacidad de memoria, por lo que es de los mejores algoritmos de optimización para el aprendizaje profundo, y es muy bueno en problemas con una gran cantidad de datos y/o parámetros.

La tabla 2.5.3 muestra las combinaciones más comunes de la función de pérdida y activación de la última capa en modelos de redes neuronales para diferentes tipos de clasificación.

Problema	**Activación de última capa**	**Pérdida**
Clasificación binaria	Sigmoide	Entropía cruzada binaria. *Binary Cross-entropy*
Multiclase	Softmax	Entropía cruzada categórica. *Categorical cross-entropy*
Regresión	Lineal	Error Medio Cuadrático MSE *Mean Squared Error*
Regresión logística	Sigmoide	MSE / Entropía Cruzada Binaria. MSE *Mean Squared Error*/*Binary cross-entropy*
Regresión	Lineal	MSE *Mean Squared Error*

Tabla 2.5.3: Función de pérdida y de activación de última capa en modelos de redes neuronales para diferentes tipos de clasificación.

Efectos de la función de optimización con diferentes tasas de aprendizaje

La figura 2.5.5 presenta información sobre la cantidad de ajuste y sobreajuste en un modelo:

La separación entre la precision de entrenamiento (línea roja) y la precisión de validación indica la cantidad de sobreajuste. La curva de error de validación en linea azul muestra una precisión de

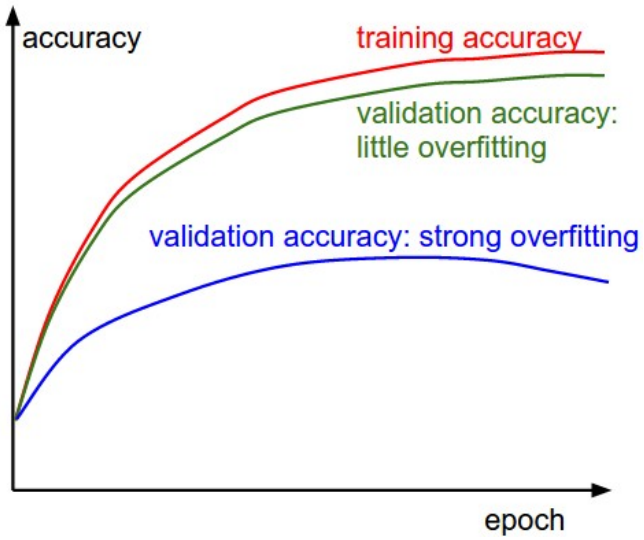


Figura 2.5.5: Función de optimización durante el entrenamiento y validación.

validación muy pequeña en comparación con la precisión de entrenamiento, lo que indica un fuerte sobreajuste. En cambio, en la curva de error de la linea verde la precisión de la validación se encuentra muy cerca a la curva de la precisión del entrenamiento, lo que muestra un pequeño sobreajuste, lo que significa que la capacidad del modelo en el entrenamiento es similar a la validación.

2.5.4.4. Optimización de la arquitectura de una Red Neuronal Artificial

La Optimización de la arquitectura de una red neuronal consiste en elegir un conjunto de hiperparámetros óptimos para un algoritmo de aprendizaje. Los hiperparámetros como se mencionó son las variables que rigen el proceso de entrenamiento que tienen que ser sintonizado para que el modelo pueda resolver de forma óptima el problema de aprendizaje automático [22]. Para una red neuronal el proceso consiste en determinar:

a). El número de capas ocultas entre la capa de entrada y la de salida.

b). La cantidad de nodos que debe tener cada capa.

c). La función de activación que se debe de utilizar.

d). La función de pérdida.

e). y las métricas para medir su rendimiento.

Decidir el número de neuronas en las capas ocultas es una parte muy importante para crear la arquitectura general de la red neuronal. Algunos casos que puede causar problemas de sobreajuste o de falta de ajuste son:

- Si se tienen pocos números de neuronas ocultas, se podría tener un gran error de entrenamiento debido a la falta de adaptación.
- Si se tiene un gran número de neuronas ocultas, se podría tener un gran error de entrenamiento debido al sobreajuste.

La optimización de la arquitectura de un modelo de ANN consiste pues en encontrar una tupla de hiperparámetros que produce un modelo óptimo que minimiza la función de pérdida de datos independientes dadas. La función objetivo toma una tupla de hiperparámetros y devuelve la pérdida asociada. La validación cruzada se utiliza para estimar esta generalización de rendimiento. El error mínimo reflejara una mejor estabilidad, y un error mayor refleja la peor estabilidad.

Métodos para determinar el número de neuronas y capas ocultas en redes neuronales artificiales ANN

Se han desarrollado varios métodos para determinar el número de neuronas ocultas. Estos métodos son de naturaleza heurística y no existe una teoría generalmente aceptada para determinar cuántas neuronas ocultas se requieren sin previamente probar durante el entrenamiento y calcular el error de generalización.

Se entiende como capacidad de generalización a la facilidad que tiene una red neuronal de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento.

En general, la red debe encontrar una representación interna que le permita generar las salidas deseadas cuando se le dan las entradas de entrenamiento, y que pueda aplicar, además, a entradas no presentadas durante la etapa de aprendizaje para clasificarlas según las características que compartan con los ejemplos de entrenamiento.

Existen algunas reglas generales, dentro de las cuales estan:

Regla de la pirámide geométrica

Esta regla se basa en la suposición de que el número de neuronas de la capa oculta ha de ser inferior al total de variables de entrada, pero superior al número de variables de salida.

Se considera que el número de neuronas de cada capa sigue una progresión geométrica, tal que, para una red con una única capa oculta el número de neuronas intermedias debe ser cercano a :

$\sqrt{V_{in} x V_{out}}$

donde:

V_{in} es el número de variables de entrada

V_{out} el total de neuronas de salida

Para dos capas ocultas se debe de calcular primeramente:

$r = \sqrt[3]{\frac{V_{in}}{V_{out}}}$

el número de neuronas en la primera capa oculta se obtiene a partir de:

$HL_1 = V_{out}\ x\ r^2$

y el número de neuronas en la segunda capa oculta está dado por:

$HL_2 = V_{out}\ x\ r$

Regla de la capa oculta-capa de entrada

Establece que el número de neuronas ocultas está relacionado con el número de neuronas de entrada. En particular suele aplicarse la regla 2x1 de forma que el número de neuronas ocultas no puede ser superior al doble del número de variables de entrada.

Regla de la capa oculta-número de patrones

Esta regla relaciona el número de neuronas de la capa oculta con el total de patrones presentados al sistema, siendo habitual la relación 1/30 (1 neurona oculta por cada 30 patrones).

Procedimiento de poda

Si bien es cierto, las reglas solo proporcionan un punto de partida, posteriormente se pueden otras técnicas como un procedimiento llamado de poda, para la cual se pueden aplicar 3 criterios:

1. A partir de una red neuronal de gran tamaño, se pueden eliminar neuronas y conexiones hasta conseguir un tamaño y precision satisfactorio.

2. Comenzar con una red neuronal pequeña e ir incrementando su tamaño añadiendo neuronas, conexiones y capas hasta conseguir el tamaño y precision satisfactoria.

3. Partir de una red de tamaño suficiente, podar las neuronas y conexiones que se consideren poco relevantes. Después se agregan neuronas con pesos aleatorios y se vuelve a entrenar la red. Este proceso se continúa hasta que se consigue un tamaño aceptable y un comportamiento satisfactorio.

2.5.4.4.1. Enfoques para optimizar hiperparámetros en una red neuronal

Existe una gran cantidad de hiperparámetros que deben ajustarse al diseñar la arquitectura de una red neuronal artificial, es un estado del arte en sí mismo encontrar la combinación correcta de estos para lograr la mayor precisión y la menor pérdida. Algunos de los enfoques generales son:

- **Búsqueda por rejilla.** Es la forma tradicional de realizar la optimización de hiperparámetros ha sido la búsqueda de rejilla o cuadricula de parámetros, consiste simplemente en una búsqueda exhaustiva a través de un subconjunto especificado manualmente del espacio de hiperparámetros de un algoritmo de aprendizaje. Un algoritmo de búsqueda por rejilla debe ser guiado por alguna medida de rendimiento, por lo general se mide por la validación cruzada en el conjunto de entrenamiento.

- **Búsqueda al azar.** La búsqueda al azar sustituye a la búsqueda exhaustiva seleccionándolas al azar. Esto puede ser aplicado a una configuración discreta aunque también se puede generalizar a espacios continuos y mixtos. Se puede aplicar especialmente cuando sólo un pequeño número de hiperparámetros afecta al rendimiento final del algoritmo de aprendizaje automático.

- **Optimización bayesiana.** La optimización bayesiana es una metodología aplicable a la optimización global de funciones de caja negra multimodales.

- **Optimización basada en el gradiente.** Se calcula el gradiente y luego busca optimizar los hiperparámetros usando descenso de gradiente. El primer uso de estas técnicas se centró en las redes neuronales, para posterior utilizarla a otros modelos como máquinas de vector soporte o de regresión logística.

- **Optimización evolutiva.** Es un algoritmo que proporciona las versiones más fuertes de una población y, generar la siguiente generación basándose en estas versiones. El proceso es iterativo, asi a partir de una población aleatoria se seleccionan las más fuertes para producir la siguien-

te generación y se detendrá una vez que la mejor solución conocida sea satisfactoria para el usuario.

2.5.4.4.2. Talos para la optimización de hiperparámetros en redes neuronales

Talos es una herramienta desarrollada en Python que permite implementar estrategias de optimización de hiperparámetros con modelos de Keras, incorporando estrategias de optimización de hiperparámetros por cuadrícula, aleatorio y probabilístico, con enfoque en maximizar la flexibilidad y eficiencia al realizar todas las combinaciones posibles en un experimento, y determinando cual es el mejor modelo de acuerdo a una métrica. Fue lanzado en mayo de 2018, su código está disponible en Github y está diseñado y probado en Python 2.+ y 3.+.

La principal ventaja de utilizar la librería de Talos, es que el código se ejecuta una vez, en lugar de ejecutarlo después de cada cambio de un solo parámetro, con esto se ahorra tiempo y facilita encontrar las mejores combinaciones con los valores de pérdida más bajos [23].

Para trabajar con Talos se requieren de 3 elementos:

1). Un diccionario hiperparámetros. Es un diccionario de Python donde se declaran los hiperparámetros y los límites que se desea incluir en el experimento.

2). Un modelo de Keras. Es un modelo de red neuronal de Keras al que se le reemplazan los parámetros que se desean incluir en el escaneo, con referencias al diccionario de hiperparámetros.

3). Un experimento de Talos. Se define el experimento usando el diccionario de hiperparámetros y el modelo de Keras definidos, al que se le agregan estrategias de optimización y de búsqueda de hiperparámetros, ya sea por cuadrícula, aleatoria o probabilística, cantidad de iteraciones, métrica, nombre del experimento, etc.

2.5.4.5. Preprocesamiento de datos

El preprocesamiento de los datos es uno de los pasos más importantes en cualquier aplicación de aprendizaje automático, ya que los datos sin procesar generalmente no se presentan en el formato adecuado, pueden llegar a ser ruidosos, que les falten algunos valores o que estos no se encuentren en la misma escala para poder modelarlos de manera correcta por lo que no resulta adecuado que sean procesados de manera directa.

Para solucionar esto ultimo, es posible aplicar técnicas ya sea de normalización o estandarización [24].

En la librería de Python Scikit-Learn existen algunos escaladores predefinidos, los más importantes son:

- StandardScaler. Elimina la media y escala los datos a la varianza unitaria. Sin embargo, los valores atípicos (*outliers*) influyen al calcular la media empírica y la desviación estándar, lo que reduce el rango de los valores de las características, por lo que no puede garantizar escalas de características equilibradas en presencia de valores atípicos.Ya que los valores atípicos en cada característica tienen diferentes magnitudes, la distribución de los datos transformados en cada característica es muy diferente, por lo que no se puede garantizar escalas de características equilibradas en presencia de valores atípicos.

- MinMaxScaler. Reescala el conjunto de datos de modo que todos los valores de las características estén en el rango [0, 1]. Al igual que con StandardScaler, MinMaxScaler es muy sensible a la presencia de valores atípicos.

- MaxAbsScaler. Es similar a MinMaxScaler excepto que los valores se asignan en el rango [0, 1]. En datos solo positivos, se comporta de manera similar a MinMaxScaler, por lo tanto, también sufre la presencia de grandes valores atípicos.

- Normalizer. Reescala el vector para que cada muestra tenga norma unitaria, independientemente de la distribución de las muestras.

2.5.5. Reducción de dimensionalidad

La dimensionalidad hace referencia al número de características o variables de entrada que se utilizan en un conjunto de datos. Cuando el número de características es muy grande en relación con el número de observaciones el entrenamiento se vuelve lento y que sea mucho más difícil encontrar una solución. Afortunadamente, existen técnicas que permiten reducir el número de dimensiones.
La reducción de dimensionalidad son un conjunto de técnicas de aprendizaje automático para el preprocesamiento de datos que permite reducir el número de variables o atributos de entrada, esencial para tareas de clasificación de datos. Estas mapean un conjunto de datos a subespacios de menor dimensión que son derivados del espacio original los cuales permiten hacer una descripción de los datos a un menor costo.
Con estas técnicas se logra acelerar el entrenamiento, convirtiendo un problema de difícil solución en

uno mucho más sencillo, sin embargo, si no se tiene el cuidado suficiente puede ocurrir que se presente alguna perdida de información lo que puede llevar a una perdida en el rendimiento de un modelo. En algunos casos inclusive puede aumentar la eficiencia, ya que al reducir la cantidad de características también se reduce el espacio de almacenamiento requerido para almacenar los datos, por lo que el compilador de Python necesitará menos tiempo para procesar el conjunto de datos.
Además de acelerar el entrenamiento, la reducción de dimensionalidad también es útil para la visualización de datos, ya que hace posible trazar un conjunto de entrenamiento de alta dimensión en un gráfico de 2 o 3 dimensiones y deducir visualmente algunos patrones.

2.5.5.1. Maldición de la dimensionalidad

La Maldición de la dimensionalidad *Curse of dimensionality* surge del proceso de analizar y organizar datos de múltiples dimensiones y representa uno de los principales obstáculos al momento de tratar de resolver problemas de optimización de aprendizaje automático.
Al aumentar la dimensionalidad, el volumen del espacio disponible aumenta exponencialmente lo que hace que los datos se vuelvan más dispersos ocasionando mayores problemas. Esta dispersión resulta problemática para cualquier método. Cuantas más dimensiones se tenga en el conjunto de entrenamiento, mayor será el riesgo de sobreajustarlo, por lo que la cantidad de datos necesarios para mantener un resultado fiable debe crecer de manera exponencial con la dimensión [10].
La maldición de la dimensión se manifiesta de dos maneras:

1. La distancia media entre los datos aumenta con el número de dimensiones.

2. La variabilidad de la distancia disminuye exponencialmente con el número de dimensiones.

Se puede observar más claramente en la figura 2.5.6, como aumenta el espacio entre los datos al pasar de 1 dimensión a una representación de 2 dimensiones y a su vez al aumentar a 3 dimensiones.

Para reducir el efecto de la maldición de la dimensión se pueden aplicar 2 técnicas:

1. Reducir el número de dimensiones utilizando técnicas de reducción de dimensionalidad.

2. Aumentar exponencialmente el tamaño del conjunto de entrenamiento para alcanzar una densidad suficiente de instancias de entrenamiento.

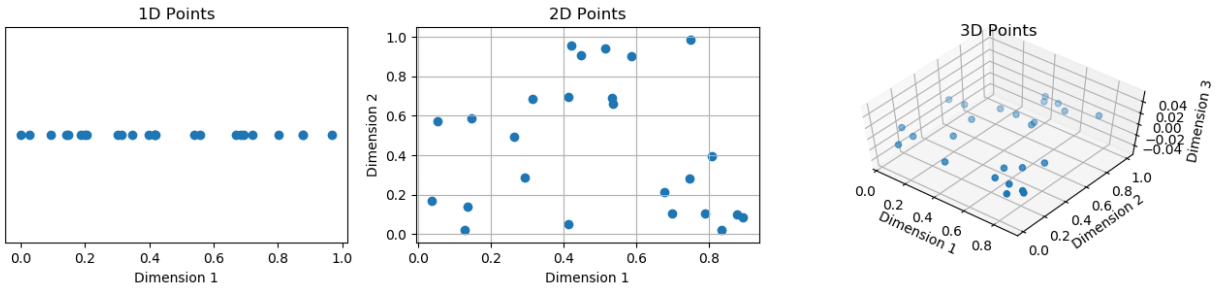


Figura 2.5.6: Maldición de la dimensionalidad. Los datos se vuelven más dispersos al aumentar el número de variables.

En la práctica, el número de instancias de entrenamiento requeridas para alcanzar una densidad dada crece exponencialmente con el número de dimensiones por lo que la cantidad de datos necesarios para mantener un resultado fiable debe crecer de manera exponencial con la dimensión. A mayor número de dimensiones se necesitan más datos para llenar el espacio. Cuando el número de dimensiones es muy alto, el espacio está casi vacío [13].
Si para un problema en particular, resulta más fácil y económico conseguir más datos, puede ser una buena opción, pero generalmente es mucho más difícil y más costoso por lo que es mucho más fácil hacer uso de las técnicas de reducción de dimensionalidad para mitigar el efecto de la maldición de la dimensión.

2.5.5.2. Ventajas de aplicar reducción de dimensionalidad

Las principales ventajas de aplicar reducción de dimensionalidad a un conjunto de datos son [25]:

- El espacio requerido para almacenar los datos se reduce a medida que disminuye el número de dimensiones.
- Menos dimensiones conducen a menos tiempo de computación / entrenamiento.
- Algunos algoritmos no funcionan bien cuando se tienen grandes dimensiones, por lo que la reducción de dimensionalidad se debe realizar para que el algoritmo sea útil.

2.5.6. Técnicas de reducción de dimensionalidad

Las técnicas de reducción de dimensionalidad tratan básicamente de capturar la mayor parte de la varianza de los datos con el menor número de dimensiones o características, ya sea manteniendo las variables más relevantes o creando un nuevo conjunto de características de los mismos datos [26].
La reducción de la dimensionalidad se realiza de 2 formas como se muestra en la figura 2.5.7 [27].
Mediante:

1. Selección de características
2. Extracción de características

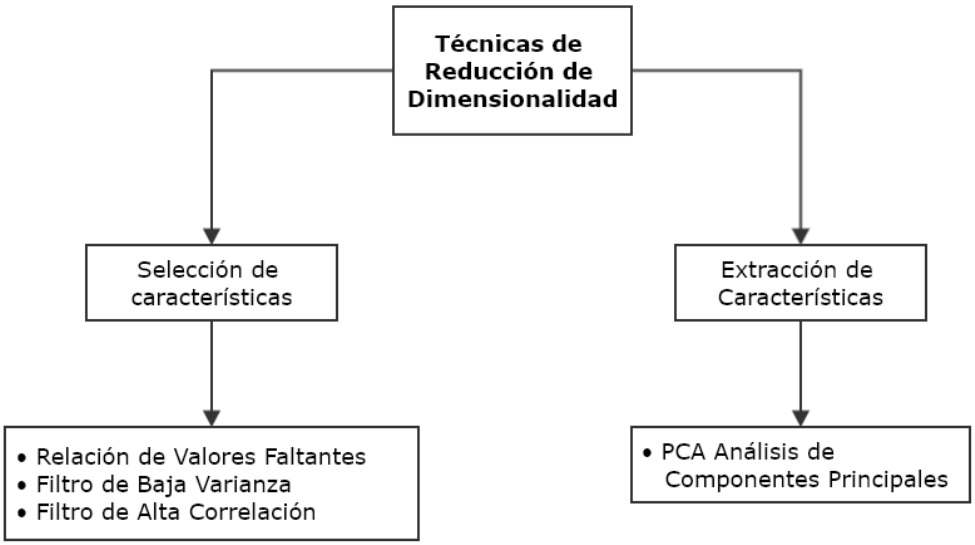


Figura 2.5.7: Técnicas para reducción de la dimensionalidad.

2.5.6.1. Selección de características

Estas técnicas mantienen solo las variables más relevantes del conjunto de datos original. La selección de características básicamente filtra características irrelevantes o redundantes del conjunto de datos.

2.5.6.1.1. Relación de valores faltantes

Se exploran los datos y si el conjunto de datos tienen una gran cantidad de valores faltantes es preferible descartar la variable ya que no contendrá mucha información, para ello se puede establecer

un valor de umbral y si el porcentaje de valores faltantes resulta mayor a dicho umbral, se elimina la variable.

2.5.6.1.2. Filtro de baja varianza

Se aplica para identificar y descartar variables constantes del conjunto de datos. Si una variable presenta prácticamente el mismo valor en todas las observaciones o muestras tendrá una baja varianza. La variable objetivo no se verá afectada por variables con baja varianza y, por lo tanto, estas variables se pueden descartar de forma segura. Para esto, se necesita calcular la varianza de cada variable y después eliminar las variables que tienen una baja varianza en comparación con las demás variables en el conjunto de datos.

2.5.6.1.3. Filtro de alta correlación

Esta técnica se utiliza para encontrar características altamente correlacionadas. Para ello se establece un umbral de alta correlación y se eliminan las características que sobrepasan dicho umbral ya que sus valores cambian de manera muy similar de manera que estas características proporcionan información redundante. Se debe de tener cuidado al establecer un valor de umbral adecuado, ya que si establece un umbral demasiado bajo, se corre el riesgo de perder información útil.

2.5.6.1.4. Importancia de características

Esta técnica asigna mayores puntuaciones a las características de entrada que tienen mayor impacto en la explicación de una observación particular en un modelo predictivo. Si se desea mejorarlo se puede centrar solo en las variables importantes eliminando variables que no son tan significativas y que tienen un rendimiento similar a las demás.

2.5.6.2. Extracción de características

Consiste en generar un conjunto de nuevas variables más pequeño, cada una de las cuales es una combinación de las mismas las cuales contienen prácticamente la misma información que las variables de entrada.

2.5.6.2.1. Análisis de componentes principales PCA

PCA *Principal Components Analysis* es de las técnicas más utilizadas para tratar datos lineales. Divide los datos en un conjunto de componentes que intentan explicar la mayor variación posible permitiendo

simplificar la complejidad de espacios muéstrales con muchas dimensiones a la vez que conserva su información [12].

Es útil como herramienta para la visualización, filtrado de ruido y extracción de características, ya que extrae un nuevo conjunto de variables de un gran conjunto de variables, estas variables extraídas se denominan componentes principales. Un componente principal es una combinación lineal normalizada. Estas nuevas características son ortogonales, lo que significa que no están correlacionadas, además se clasifican según su varianza explicada.

El primer componente principal PC1 es una combinación lineal de variables que captura y determina la dirección de mayor varianza dentro del conjunto de datos original. Cuanto mayor sea la varianza capturada en el primer componente, mayor será la información capturada por este componente. Ningún otro componente puede tener una varianza más alta que el primer componente principal.

El segundo componente principal PC2 también es una combinación lineal que captura la varianza restante en el conjunto de datos y no está correlacionada. La correlación entre el primer, segundo y demás componentes debe de ser cero, y si los componentes no están correlacionados, sus direcciones deben ser ortogonales. El ejemplo se muestra en la figura 2.5.8.

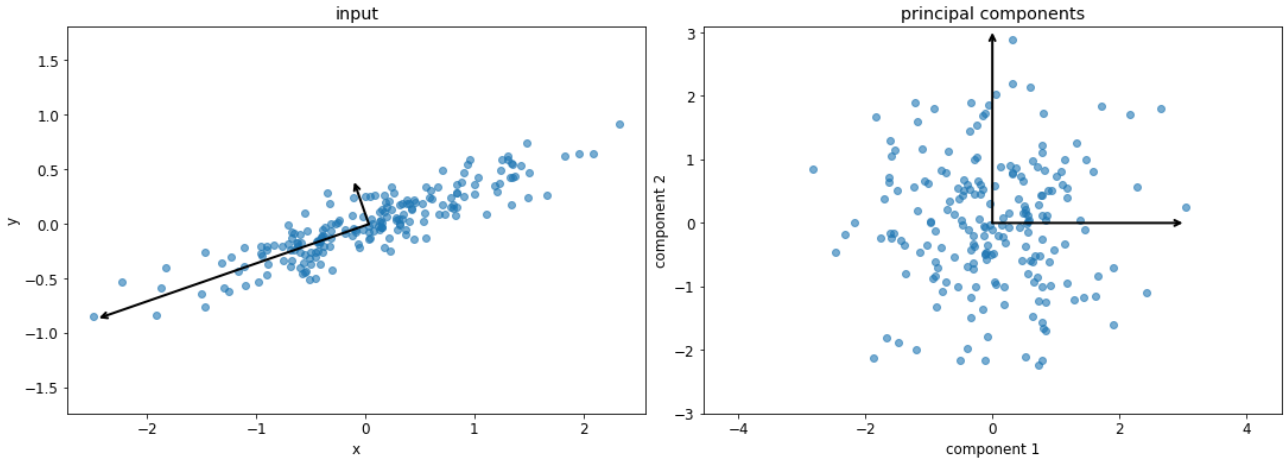


Figura 2.5.8: Análisis de componentes principales PCA. Se muestran los 2 primeros componentes.

Los demas componentes principales siguen el mismo concepto, capturan la variación restante sin correlacionarse con el componente anterior, asi puede reducirse la dimensionalidad limitando el número de componentes principales de acuerdo a la varianza explicada acumulada.

Es importante mencionar que siempre se debe normalizar el conjunto de datos antes de realizar PCA porque la transformación depende de la escala, si no se realiza, las características que están en la es-

cala más grande dominarían los nuevos componentes principales.

Los pasos para implementar PCA son:

1. Estandarizar o normalizar los datos.
2. Calcular la matriz de covarianza (con dimensión d).
3. Obtener los vectores propios y los valores propios de la matriz de covarianza.
4. Ordenar los valores propios en orden descendente y elegir los k vectores propios que correspondan a los valores propios más grandes,donde k es el número de dimensiones en el nuevo subespacio de entidad .
5. Construir la matriz de proyección W a partir de los k vectores propios seleccionados.
6. Transformar el conjunto de datos original X multiplicando por W para obtener un subespacio k-dimensional de características Y.

2.5.7. Selección de métricas de clasificación

Para determinar qué modelo es el mejor se debe elegir la métrica de selección sobre la cual se califican los diferentes modelos de aprendizaje automático. Las principales métricas de clasificación usadas en aprendizaje automático son:

1. Matriz de confusión y las tasas y métricas obtenidas a partir de esta.
2. Pérdida logarítmica.
3. Coeficiente de determinación R^2.
4. Área bajo la curva de funcionamiento del receptor AUC / ROC.

2.5.7.1. Matriz de confusión

Una matriz de confusión es una medida de rendimiento de un algoritmo de aprendizaje automático, generalmente de un modelo de clasificación de aprendizaje supervisado. Se representa por medio

de una tabla de datos de prueba para los que se conocen los valores verdaderos y proporciona información no solo de los errores en los resultados del modelo, también del tipo de errores (figura 2.5.9).

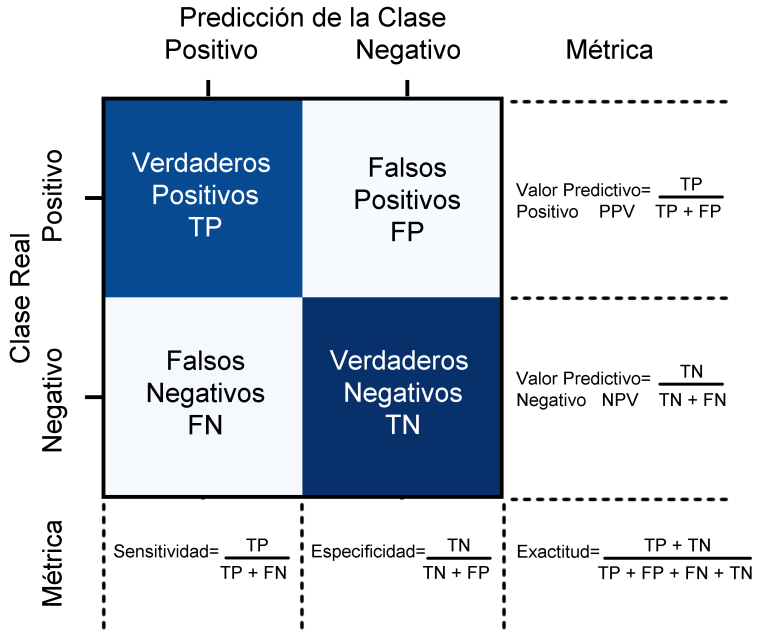


Figura 2.5.9: Métricas a partir de la matriz de confusión para un clasificador binario.

Los elementos diagonales representan el número de aciertos para los cuales la etiqueta pronosticada es igual a la etiqueta verdadera, mientras que los elementos fuera de la diagonal son aquellos que están mal etiquetados por el clasificador [10].
Cuanto mayores sean los valores diagonales de la matriz de confusión, mayores predicciones correctas. En ocasiones se realiza una normalización de los valores, ya que esto muestra más fácilmente si existe un desequilibrio en las muestras tomadas de cada clase a fin de tener una interpretación más visual de qué clase se está clasificando erróneamente.

Dado un modelo de aprendizaje automático y una instancia, existen 4 resultados posibles [28]:

1. Verdadero Positivo TP. Un verdadero positivo es un resultado en el que el modelo predice correctamente la clase positiva, y ocurre cuando la instancia es positiva y se clasifica como positiva.

2. Verdadero Negativo TN. Un verdadero negativo es un resultado en el que el modelo predice correctamente la clase negativa, es decir, cuando la instancia es negativa y se clasifica como negativa.

3. Falso Positivo FP. Un falso positivo es un resultado en el que el modelo predice incorrectamente la clase positiva, y ocurre cuando la instancia es negativa y se clasifica como positiva. Se le conoce como error tipo I.

4. Falso Negativo FN. Un falso negativo es un resultado en el que el modelo predice incorrectamente la clase negativa, es decir, cuando la instancia es positiva y se clasifica como negativa. Conocido como error de tipo II

Tasas y métricas obtenidas a partir de la matriz de confusión.

Accuracy - Exactitud: La exactitud es una medida para evaluar los modelos de clasificación. Indica con qué frecuencia es correcto el clasificador.

$$Accuracy = \frac{numero\_predicciones\_correctas}{numero\_total\_predicciones}$$

Para la clasificación binaria, la precisión también se puede calcular en términos de positivos y negativos

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

Precision - Precisión: La precisión es la capacidad de un modelo de clasificación para devolver solo instancias relevantes. Indica de los positivos pronosticados cuántos de ellos son realmente positivos.

$$Precision = \frac{TP}{(TP + FP)}$$

Recall - Recuperación, Sensitivity - Sensibilidad o TPR: Es el número de verdaderos positivos dividido entre el número de verdaderos positivos más el número de falsos negativos.
Los verdaderos positivos son datos clasificados como positivos por el modelo cuando la instancia es positiva, lo que significa que son correctos, y los falsos negativos los cuales son los datos que el

modelo identifica como negativos que en realidad son positivos, es decir, que se clasifican incorrectamente. También se conoce como Tasa de Verdaderos Positivos (True Positive Rate TPR), los cuales representan la proporción de casos positivos que fueron correctamente identificadas por el algoritmo.

$$Recall(Sensitivity\, o TPR) = \frac{TP}{(TP + FN)}$$

Especificity-Especificidad o TNR: Es la proporción de casos verdaderamente negativos que se clasificaron como negativos; es decir, es una medida de qué tan bien identifica los casos negativos. También se le conoce como Tasa de Verdaderos Negativos, (True Negative Rate TNR). Se trata de los casos negativos que el algoritmo ha clasificado correctamente.

$$Especificity(TNR) = \frac{TN}{(TN + FP)}$$

F1-Score. Esta métrica básicamente resume la precisión y sensibilidad en una sola métrica. Se utiliza cuando se desea buscar un equilibrio entre Precision y Recall. Determina el promedio ponderado de precisión y recuperación.
Esta métrica tiene en cuenta tanto los falsos positivos como los falsos negativos, por ello es de gran utilidad cuando la distribución de las clases es desigual. Se obtiene a partir de:

$$F1Score = 2\, x \frac{Recall\, x\, Precision}{Recall + Precision}$$

Classification report - Informe de clasificación: La función de informe de clasificación *classification_report* del módulo de Python sklearn crea un informe que muestra las principales tasas y métricas de clasificación. Este reporte incluye:

- *f1_macro macro-averaged.* Macropromedio. Promedio de la media no ponderada por etiqueta.
- *f1_micro micro-averaged.* Micropromedio. Promedio del total de verdaderos positivos, falsos negativos y falsos positivos.

- *f1_weighted weighted average.* Promedio ponderado. Promedio de la media ponderada de soporte por etiqueta.
- *f1_samples.* Promedio de la muestra, para clasificación de múltiples etiquetas. Solo se muestra para etiquetas múltiples o clases múltiples con un subconjunto de clases.

2.5.7.2. Pérdida Logarítmica

La pérdida logarítmica también llamada pérdida de regresión logística o pérdida de entropía cruzada, mide el rendimiento de un modelo de clasificación donde la entrada de predicción es un valor de probabilidad entre 0 y 1. El objetivo del modelo de aprendizaje automático es minimizar este valor [29]. Un modelo perfecto tendría una pérdida logarítmica de 0 y aumenta a medida que la probabilidad prevista difiere de la etiqueta real. Para un buen algoritmo predictivo la pérdida debe estar cerca de cero. El objetivo de un modelo de aprendizaje automático es minimizar este valor.

2.5.7.3. Coeficiente de determinación R^2

El coeficiente de determinación, conocido también como R^2 es otra métrica utilizada para evaluar un modelo, está estrechamente relacionado con el Error Medio Cuadrático MSE *Mean Square Error*, tiene la ventaja de no tener escala. No importa si los valores de salida son muy grandes o muy pequeños, el valor de R^2 siempre va a estar entre $-\infty$ y 1. Cuando R^2 es negativo, significa que el modelo es peor que predecir la media, un valor cercano a cero indica un modelo muy cercano a la media o línea-base y un valor cercano a 1 indica un modelo con un error cercano a cero.
Matemáticamente se define como el porcentaje de variación en la variable de respuesta que es explicado por un modelo lineal. Es decir:

$$R^2 = \frac{Variacion\_explicada}{Variacion\_total}$$

En conclusión, R^2 es la relación entre lo bueno que es un modelo y lo bueno que es el modelo de línea-base.

$$R^2 = 1 - \frac{MSE(modelo)}{MSE(linea\_base)}$$

Donde:

$$MSE(modelo) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$MSE(linea\_base) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y})^2$$

Donde:

y_i es la salida esperada real

$\hat{y}$ es la predicción del modelo

$\overline{y}$ es la media de la y observada

2.5.7.4. Curva AUC/ROC

La curva AUC/ROC *Area Under Curve - Receiver Operating Characteristics* Área Bajo la Curva/-Características de Funcionamiento de Receptor es otra de las métricas de evaluación para verificar el rendimiento de cualquier modelo de clasificación. Esta considera todos los umbrales de clasificación posibles de una curva ROC. El área bajo la curva ROC es la probabilidad de que un clasificador tenga más seguridad de que una muestra positiva elegida al azar sea realmente positiva con respecto a que una muestra negativo elegida al azar sea positiva [10].

Se representa por medio de un diagrama que ilustra la capacidad de diagnóstico de un sistema clasificador binario a medida que varía su umbral de discriminación señalando cuándo el modelo es capaz de distinguir entre clases. A mayor AUC, mejor será el modelo para predecir valores 0's como pertenecientes a la clase 0 y 1's a la clase 1.

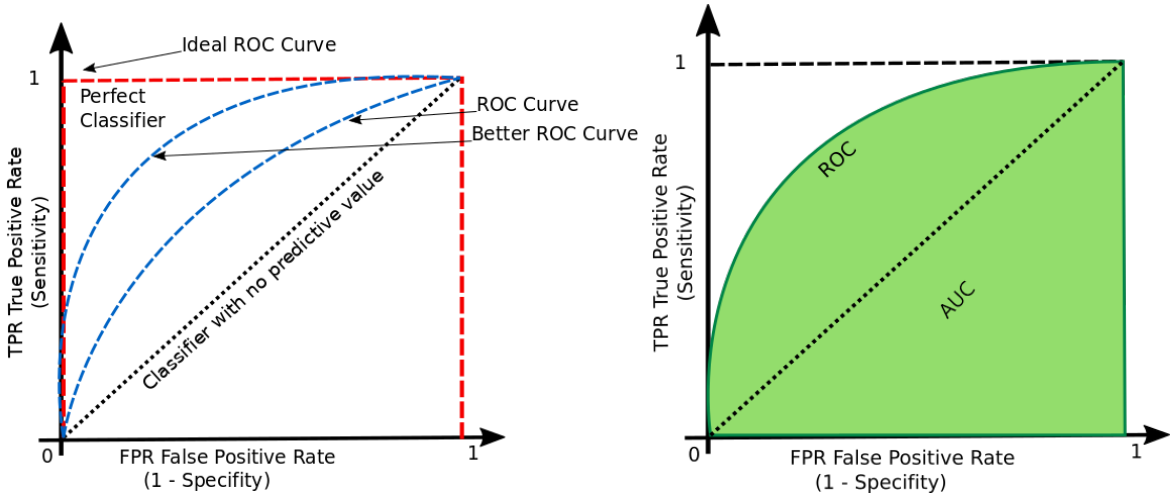
Esta curva resulta de comparar la sensibilidad frente a la especificidad para un sistema de clasificación binaria según se varía el umbral de discriminación, representa la relación de la tasa de verdaderos positivos TPR contra la tasa de falsos positivos FPR (figura 2.5.10.a).

Tasa de Verdaderos Positivos TPR. Es sinónimo de exhaustividad, se define como:

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

Tasa de Falsos Positivos FPR. Se define a partir de:

$$FPR = \frac{FP}{FP + TN}$$



(a) Curva ROC: Tasa de TPR frente a FPR para diferentes umbrales de clasificación.

(b) Area Bajo la Curva AUC/ROC

Figura 2.5.10: Curva AUC/ROC.

A la tasa de verdaderos positivos se le conoce como sensibilidad, recall o probabilidad de detección, mientras que a la tasa de falsos positivos se le conoce como probabilidad de falsa alarma y se puede calcula como 1 – Especificidad.

Como se presenta en la figura 2.5.10.b AUC mide el área por debajo de la curva ROC de (0,0) a (1,1), y puede tomar un valor que oscila entre 0 y 1. Un modelo cuyas predicciones son 100 % incorrectas tiene un AUC de 0.0, si las predicciones son 100 % correctas tiene un AUC de 1.0.

Capítulo 3

RECURSOS DE SOFTWARE

3.1. Herramientas utilizadas

3.1.1. Linux Mint 19

Linux Mint 19 Tara es una distribución de GNU/Linux basada en Ubuntu, a su vez en Debian desarrollada por el equipo de Linux Team y su comunidad. Se basa en Ubuntu 18.04 LTS Bionic Beaver y cuenta con el Kernel 4.15, tiene soporte a largo plazo compatible hasta el año 2023. Es un sistema operativo actualizado y muchas características cómodas, poderosas y fáciles de usar. Soporta varios formatos al incluir software propietario y empaquetado con una variedad de aplicaciones gratuitas y de código abierto. Además, dependiendo de la plataforma de Linux Mint que se elija, se encuentra en diferentes sabores como Cinnamon 3.8, MATE 1.20 y Xfce 4.12 [30].

3.1.2. Python 3.7

Es un lenguaje de programación de código abierto multiparadigma, que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Python es un lenguaje interpretado, escalable, robusto y con una sintaxis que favorece el código legible, que permite ahorrar tiempo y recursos, lo que facilita su comprensión e implementación [31].
Se puede aplicar fácilmente a la Inteligencia Artificial, además, existen bibliotecas que se usan mucho en los algoritmos de machine learning como Scikitlearn, una biblioteca gratuita de aprendizaje automático que presenta varios algoritmos de regresión, clasificación y agrupamiento. La versión 3.7 de Python tiene muchas características nuevas, las cuales se pueden encontrar en la documentación

oficial.

La demanda de usuarios de Python crece exponencialmente cada año. De acuerdo al informe de la web de referencia para desarrolladores de todo el mundo web Stackoverflow la cual presenta la evolución del volumen de búsquedas por tipo de lenguaje, señala claramente que Python está a la cabeza.

En el ranking TIOBE del mes de Diciembre de 2020 mostrado en la figura 3.1.1 Python (línea verde en la gráfica) ha escalado a la segunda posición, solo detrás de C. TIOBE es un índice de la comunidad de programación que mide la popularidad de los lenguajes de programación. El índice se actualiza una vez al mes. Las calificaciones se basan en la cantidad de ingenieros calificados en todo el mundo, cursos y proveedores externos, y se calcula a partir del número de resultados del motor de búsquedas para consultas que contienen el nombre de un lenguaje de programación y cubre búsquedas en Google, Google Blogs, MSN, Yahoo!, Baidu, Wikipedia y YouTube [32].

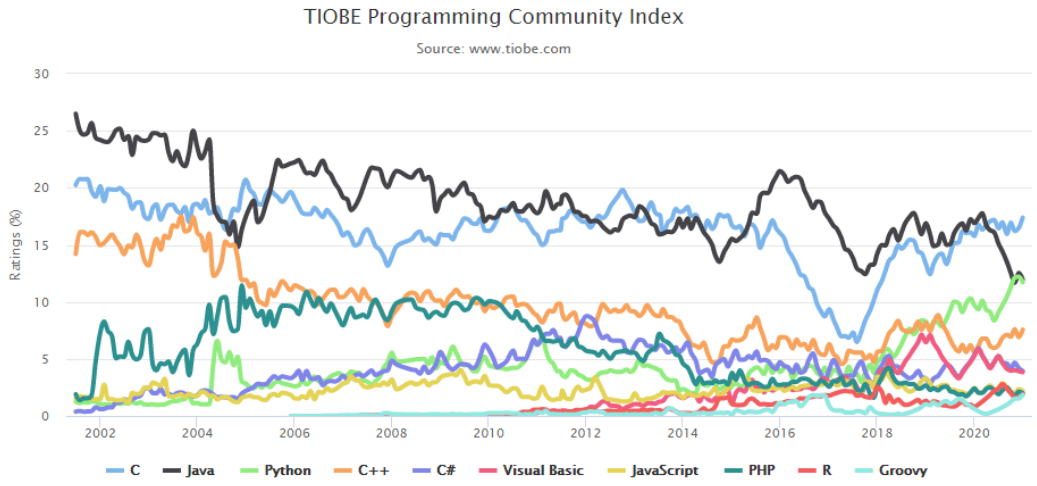


Figura 3.1.1: Ranking TIOBE de Diciembre de 2020.

3.1.3. Bibliotecas y herramientas de Python escenciales utilizadas en Aprendizaje Automático

3.1.3.1. NumPy

NumPy es una librería fndamental para la computación científica en Python. Es una extensión open-source que agrega mayor soporte para vectores y matrices multidimensionales. Es una biblioteca de funciones matemáticas de alto nivel para operar cualquier tipo de problema matemático como operaciones de álgebra lineal, transformada de Fourier y generadores de números pseudoaleatorios.

En Scikit-Learn, la matriz NumPy es la estructura de datos fundamental, cualquier dato que esté usando tendrá que ser convertido a una matriz NumPy.

3.1.3.2. SciPy

SciPy es otra biblioteca libre y de código abierto que proporciona una colección de funciones para computación científica en Python, entre las cuales hay funciones de rutinas avanzadas de álgebra lineal, optimización de funciones matemáticas, procesamiento de señales, funciones matemáticas especiales y distribuciones estadísticas.

3.1.3.3. Scikit-Learn

Es otra biblioteca de software libre de aprendizaje automático que está construida sobre NumPy y SciPy y cuenta con varios algoritmos de clasificación, regresión y agrupación, incluyendo máquinas de vectores de soporte, bosques aleatorios, aumento de gradiente, k-means, entre otros.

3.1.3.4. Pandas

Está construido alrededor de un marco de datos, es decir, una estructura de datos en forma de tabla similar a una tabla de Excel que proporciona una gran variedad de métodos para manipular y modificar tablas numéricas y series temporales. Permite realizar consultas de SQL y a diferencia de NumPy que requiere que todas las entradas en una matriz sean del mismo tipo, Pandas permite que cada columna tenga un tipo distinto, ya sea de números enteros, fechas, números de punto flotante y cadenas. Tiene la capacidad de manejar una gran variedad de formatos de archivo y bases de datos, como SQL, archivos de Excel y archivos CSV.

3.1.3.5. Matplotlib

Matplotlib es la principal biblioteca para la generación de gráficos científicos en Python. Proporciona funciones para realizar visualizaciones como gráficos de líneas, histogramas, diagramas de dispersión, etc.

3.1.3.6. Keras

Keras es una API de alto nivel de aprendizaje profundo de redes neuronales de código abierto escrita en Python que se ejecuta sobre la plataforma de aprendizaje automático *TensorFlow*. Fue desarrollado con el enfoque de permitir una experimentación rápida y fue concebida para actuar como una interfaz en lugar de ser una *framework* de aprendizaje profundo *standalone*.

3.1.3.7. Anaconda

Anaconda es una plataforma de distribucion libre y de codigo abierto de lenguaje Python y R, utilizada en ciencia de datos y aprendizaje automatico. Es multiplataforma y posee un conjunto de herramientas que contiene Conda, Anaconda Navigator, Python y cientos de paquetes cientificos [33]. Conda funciona con una interfaz de línea de comandos, mientras que Anaconda Prompt en Windows y terminal en Linux, por su parte Navigator es una interfaz gráfica de usuario de escritorio que permite iniciar aplicaciones y administrar fácilmente paquetes, entornos y canales de conda sin usar comandos en línea de comandos. Contiene Cuadernos de Jupyter (*Jupyter Notebooks*) e IDE´s más populares [28].

3.1.3.8. Jupyter Notebook

Jupyter Notebook es un IDE *Integrated Development Environment* Entorno de Desarrollo Integrado de código abierto que proporciona Anaconda basado en la web que permite crear y compartir documentos que pueden contener código, ecuaciones, imágenes y texto, se pueden compartir fácilmente y su código puede producir una salida rica e interactiva, que incluye HTML, imágenes y videos.
El formato de Jupyter Notebook ipynb se ha convertido en un estándar y se puede renderizar en múltiples IDE, GitHub y otras plataformas. Dentro de sus principales características están:

- Admite más de 40 lenguajes de programación, incluidos Python y R.
- Permite compartir los archivos mediante correo electrónico, Dropbox, GitHub y Jupyter Notebook Viewer.
- Salida interactiva.
- Su código puede producir una salida rica e interactiva: HTML, imágenes, videos y código en LaTeX.

- Integración de big data, como Apache Spark, de Python y R. Es posible explorar esos mismos datos con pandas, scikit-learn, ggplot2 y TensorFlow.

3.1.3.9. Spyder

Spyder es un IDE gratuito y de código abierto para Python, presenta funciones avanzadas de edición, análisis, depuración y creación. Posee herramientas de desarrollo integral como la exploración de datos, ejecución interactiva, inspección y diferentes capacidades de visualización.

3.1.4. Bibliotecas de Python para el procesamiento de registros sísmicos

3.1.4.1. ObsPy

ObsPy es un proyecto de código abierto dedicado a proporcionar herramientas de Python para procesar datos sísmicos [34]. Posee rutinas para el análisis de series de tiempo de registros para los formatos más comunes como .seed, .mseed, .sac, entre otros y de clientes para acceder a centros de datos y rutinas de procesamiento de señales que permiten la manipulación de series de tiempo [35]. Además soporta la lectura, escritura y conversión de formatos como SEED / MiniSEED, XML-SEED, GSE2 y SAC [36]. Posee rutinas que permiten eliminar la respuesta instrumental,realizar un análisis frecuencial, aplicar filtros y graficar las trazas.

Representación interna de datos de forma de onda en ObsPy

En ObsPy, los datos de forma de onda están representados por objetos *Stream* que actúan como un contenedor de objetos *Trace*. Los objetos *Trace* contienen una ventana de tiempo única, contigua e igualmente muestreada de datos de forma de onda junto con su metainformación asociada, como los atributos *data*, la cual es una matriz unidimensional *NumPy* y el atributo *stats* el cual contiene información adicional de estadísticas.

El atributo *stats* almacena cuatro identificadores SEED: *network*, *station*, *location* y *channel*, que brinda la ubicación física del instrumento de grabación. Además, contiene los tiempos de la primera y última muestra, frecuencia, intervalo de muestreo y número de muestras. El atributo *endtime* es de solo lectura y se ajusta automáticamente si la hora de inicio, frecuencia de muestreo o el número de muestras en la matriz ha cambiado.

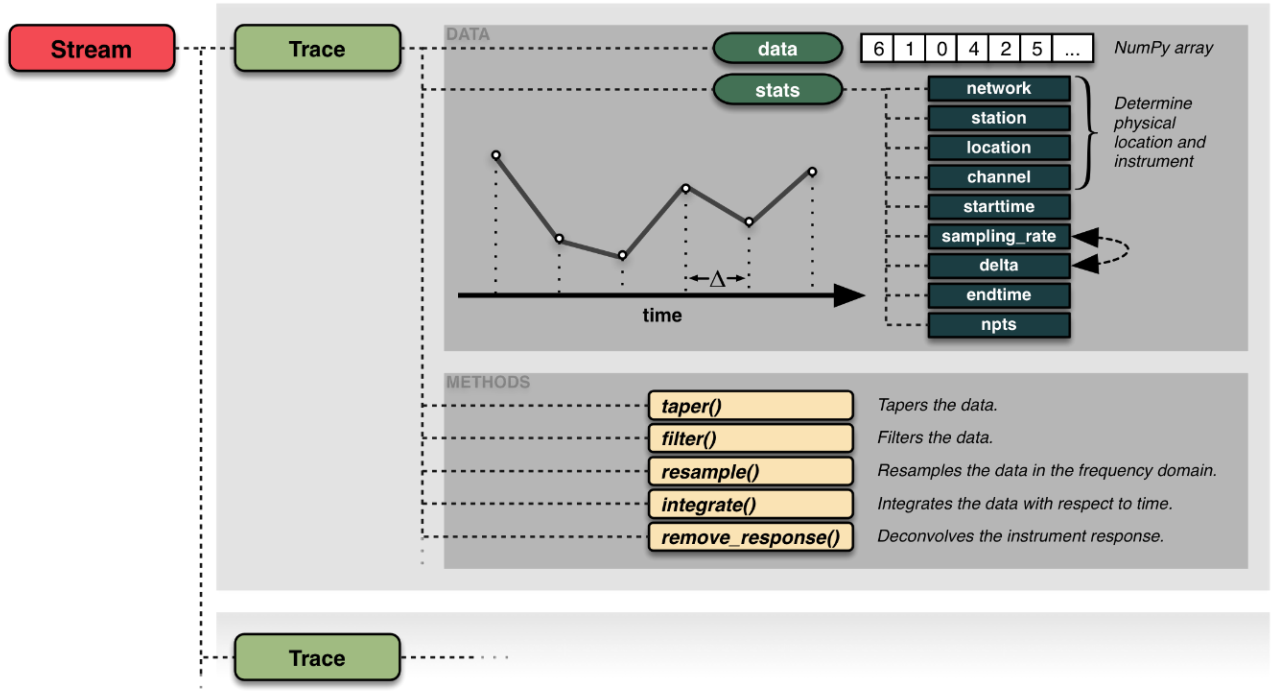


Figura 3.1.2: Representación interna de datos de forma de onda de ObsPy.

ObsPy también detecta automáticamente el formato de archivo y carga los datos de objetos *stream* que puede contener múltiples trazas. Los formatos SEED, MiniSEED y GSE2, pueden contener múltiples trazas en un solo archivo. Estos registros de datos pueden ser leídos cada uno como un objeto *trace*. Los atributos de encabezado de la primera traza (tr = st [0]) pueden ser direccionados por tr.stats (por ejemplo, tr.stats.sampling_rate). El atributo tr.data contiene los datos en un objeto numpy.ndarray, por lo que los datos pueden ser procesados por Python estándar, NumPy, SciPy, matplotlib o rutinas de ObsPy [37].

Los registros generalmente se encuentran en formato SEED el cual es un formato diseñado para archivar e intercambiar series de tiempo sismológicas y metadatos. Para el análisis de las señales sísmicas es necesario convertirlos de formato a SAC el cual es el formato desarrollado para analizar datos en series de tiempo. Aqui las trazas se escriben por separado y poseen un encabezado con la información de la red, estación, datos, etc.

Capítulo 4

METODOLOGÍA

La metodología aplicada (figura 4.0.1 y 4.0.2) fue la siguiente:

1. Configurar el entorno de trabajo.

2. Importar bibliotecas y módulos.

3. Obtener los datos sísmicos y cambiarlos de formato.

4. Realizar el preprocesamiento de la señal sísmica. Seleccionar el canal vertical y realizar la corrección instrumental, eliminación de la tendencia en los datos, filtrado y normalización.

5. Definir y extraer las características. Definir las características en el dominio del tiempo (9 características) y de la frecuencia (11 características), dividir la señal en 6 segmentos, y aplicar la extracción de características para obtener un vector de 120 características por cada registro formando el *dataset* de trabajo.

6. Experimentación:
   Se definen 2 experimentos: con la entrada de 120 características y realizando análisis de reducción de dimensionalidad para formar un nuevo modelo de red neuronal. y se realizarán los pasos 7 al 11 con cada uno de los experimentos.

7. Definir arquitectura del modelo de red neuronal:

   a) Leer el *dataset* y normalizar. Definir conjunto de datos de entrenamiento (70 %) y de prueba (30 %).

b) Definir los límites de búsqueda en el diccionario de hiperparámetros.

c) Construir el modelo de red neuronal de Keras con referencias al diccionario de Talos.

d) Configurar parámetros del experimento.

e) Ejecutar experimento de Talos con el comando *Scan()*.

f) Análizar los informes y gráficas obtenidas.

g) Evaluar el modelo.

h) Implementar el modelo.

8. Restauración del modelo de red neuronal.

9. Cargar modelo y analizar resultados.

10. Informe de las principales tasas y métricas de clasificación.

    a) Errores de clasificacion.

    b) Matriz de confusión.

    c) Perdida logarítmica.

    d) Coeficiente de determinación.

    e) Curva ROC/AUC.

11. Arquitectura y rendimiento del modelo obtenido.

    a) Gráfica del rendimiento del modelo de Red Neuronal Artificial.

    b) Gráfica de pérdidas del modelo de Red Neuronal Artificial.

    c) Gráfica de errores de clasificación del modelo.

Al final se evaluarán los resultados con ambos experimentos mediante la comparación de los resultados y determinar el optimo.

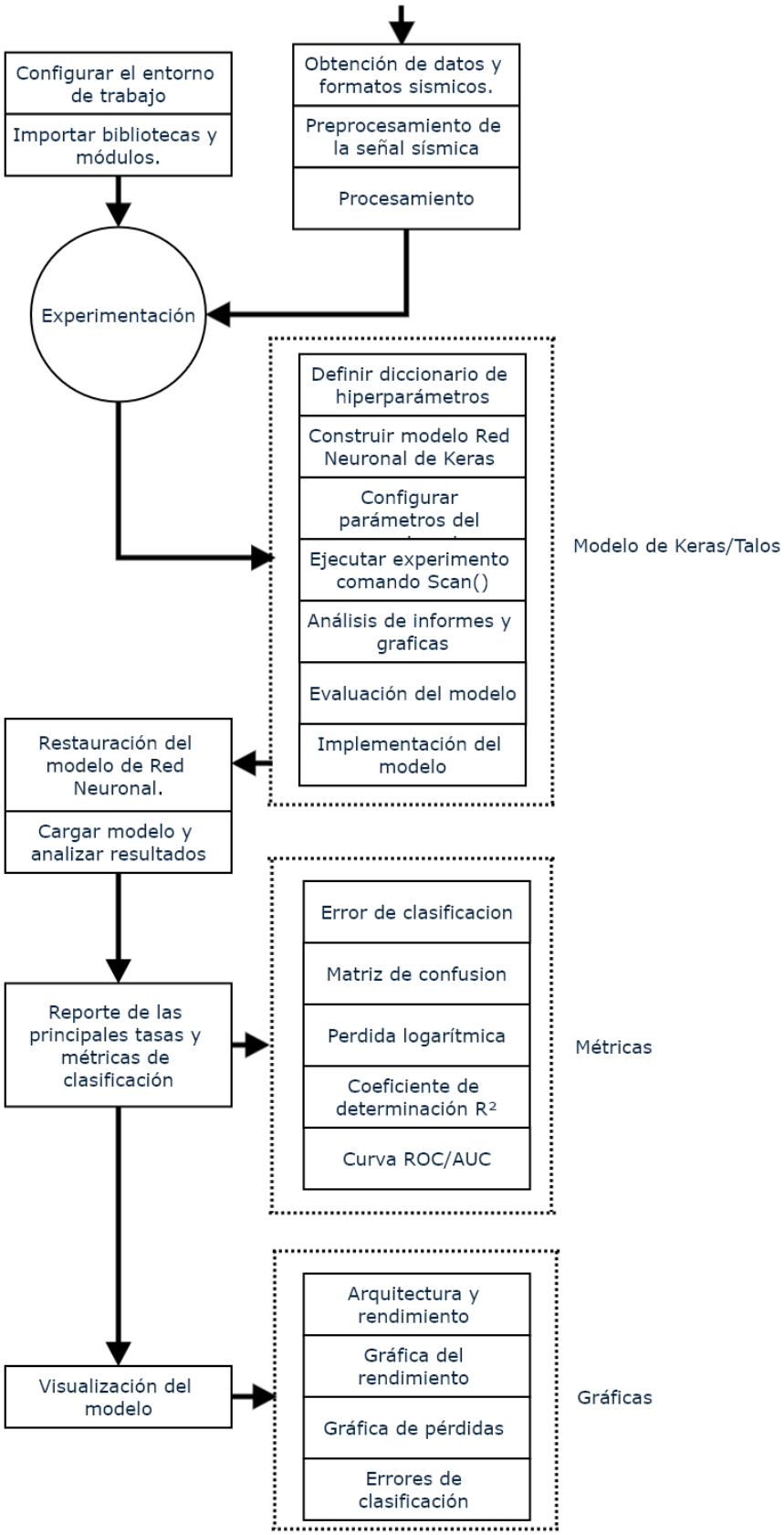


Figura 4.0.1: Metodología utilizada para la Red Neuronal

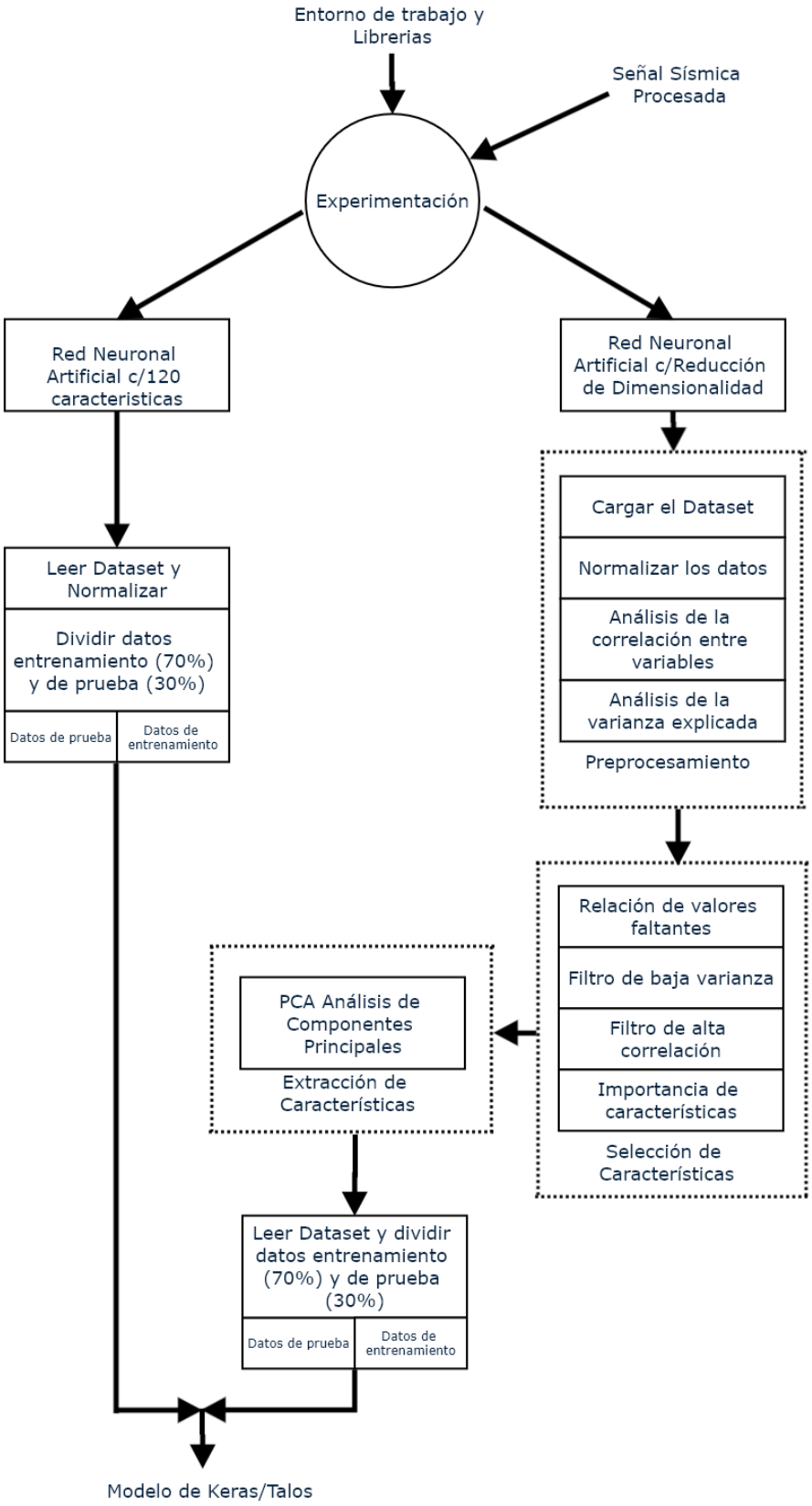


Figura 4.0.2: Experimentos realizados de Talos con los modelos de Keras.

4.1. Configurar entorno de trabajo

Los experimentos se ejecutaron en una computadora portátil Sony Vaio con procesador Intel Core i5 a 1.8GHz x 2, RAM 5.7Gb, Disco duro 342Gb, Tarjeta gráfica Card: Intel 3rd Gen Core processor Graphics Controller, con un sistema operativo Linux Mint 19 Cinnamon y además se instala Anaconda3 Individual Edition (figura 4.1.1).

Figura 4.1.1: Entorno de trabajo.

4.2. Importar bibliotecas y módulos

Anaconda3 Individual Edition incluye Python 3.7+, 1.5.2, NumPy 1.19.2, Pandas 1.1.3 y Matplotlib 3.3.2. Además se requiere instalar las herramientas: Keras 2.3.1 y Obspy 1.2.2

4.3. Obtención de datos y formatos sísmicos

Los datos están disponibles en los principales centros de datos y formatos sísmicos como IRIS-DMC *Incorporated Research Institutions for Seismology-Data Management Center* y en el SCEDC *Southern California Earthquake Data Center* (figura 4.3.1).

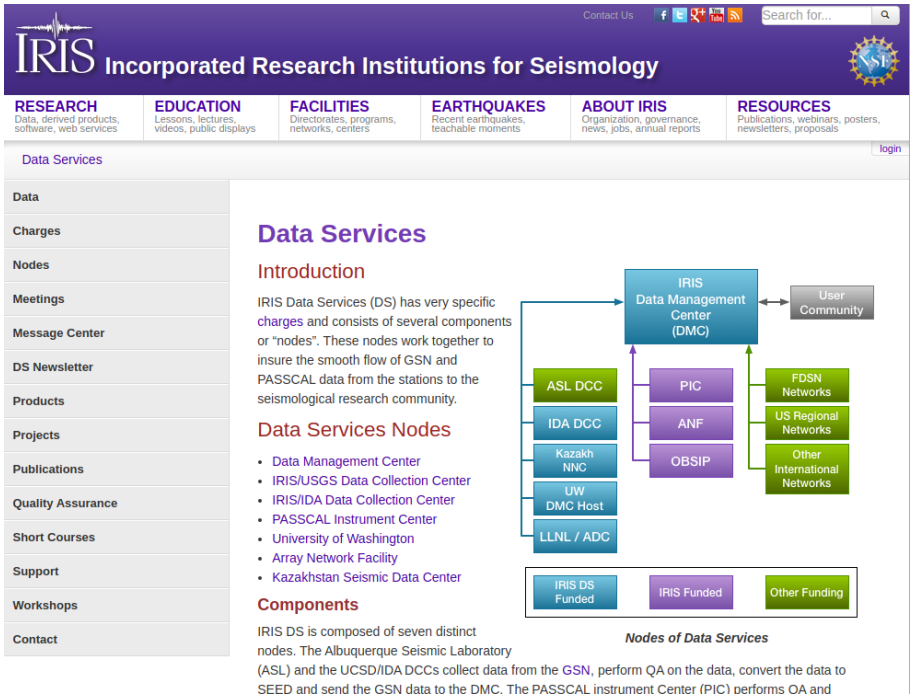


Figura 4.3.1: IRIS-DMC *Incorporated Research Institutions for Seismology- Data Management Center.*

4.3.1. Formatos de registros sísmicos

SEED

(*Standard for the Exchange of Earthquake Data* - Estándar para el intercambio de datos sismológicos) es un formato estándar internacional para el intercambio de datos sismológicos digitales entre instituciones de datos sin procesar. El manual de referencia de SEED versión 2.4 es publicado por la FDSN *Federation of Digital Seismographic Networks* Federación de Redes Sismográficas Digitales, IRIS *Incorporated Research Institutions for Seismology* Instituciones de Investigación Incorporadas para la Sismología y el USGS *United States Geological Survey* Servicio Geológico de Estados Unidos. Es un formato para datos digitales medidos en intervalos de tiempo iguales [38].

Mini-SEED Es una versión reducida de SEED sin la información del encabezado.

SAC (*Seismic Analysis Code* Código para Análisis Sísmico) Desarrollado para analizar datos en series de tiempo, especialmente sismicos. Es uno de los formatos y programas de datos más utilizados en la comunidad de investigación sismológica. El código fuente de SAC está disponible públicamente

para instituciones afiliadas a IRIS.

El formato de datos estándar para SAC es binario. Un archivo SAC binario contiene datos ASCII de longitud fija (encabezado), que describen los datos posteriores en formato binario con longitud variable. Para los datos sísmicos, esto significa un solo componente de datos registrado en una sola estación sísmica.

El encabezado SAC consta de información de cada dato, como intervalo de muestreo, hora de inicio, duración, ubicación de la estación, ubicación del evento, componentes, etc. Varios comandos de SAC utilizan estos parámetros para procesar los datos, por lo que resulta importante mantener la información del encabezado completa y actualizada.

4.3.2. Preprocesamiento de datos sísmicos para aprendizaje Autómatico

Para extraer las características de las series de tiempo es necesario realizar algunos procesos descritos en la figura 4.3.2.

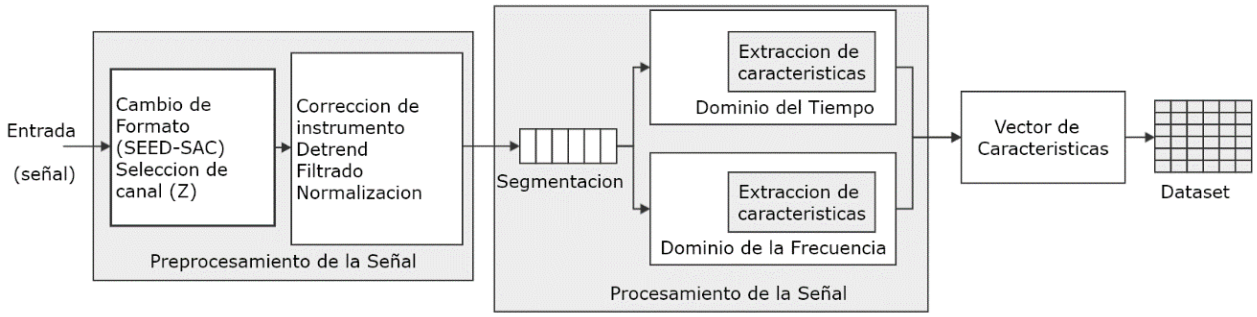


Figura 4.3.2: Procesamiento de la señal sísmica: preprocesamiento y extracción de características.

4.3.2.1. Cambio de formato y selección de canal

La frecuencia de muestreo del sensor utilizado (sensor de banda ancha Streckeisen STS-2) es de 20Hz y las componentes de las señales sísmicas a analizar poseen componentes de 1 hasta 10 Hz, por lo que se cumple con la condición Nyquist que establece que la frecuencia de muestreo debe ser mayor o igual al doble de la frecuencia máxima de la señal [4], entonces se procede a leer las trazas, se convierten de formato de .seed a sac, y se seleccionan los canales verticales BHZ.

4.3.2.2. Corrección de instrumento

Una vez que se registran los datos sísmicos, se elimina la respuesta del instrumento mediante la deconvolución de la función de transferencia, normalmente parametrizada en polos y ceros.

4.3.2.3. Corrección de línea base

La corrección de línea-base o *Detrend* elimina el desplazamiento constante, lineal, polinómico o curvo de los datos cuando los valores se encuentran desplazados respecto a la línea cero a lo que se le conoce como tendencia. Generalmente ocurre cuando el sensor no se encuentra nivelado en su emplazamiento, aunque también puede deverse al sistema de registro.

4.3.2.4. Filtrado

El ruido sísmico es un término general aplicado a cualquier tipo de señal no deseada en los datos sísmicos que no producen información o información falsa que no se puede utilizar en el proceso. La función de un filtro es eliminar las partes no deseadas de la señal (ruido sísmico) o extraer partes útiles (componentes en una determinada banda de frecuencias).
La principal razón detrás de los ruidos son las perturbaciones indeseables de la señal mientras se acumulan datos de los sismómetros. Las fuentes de estos ruidos pueden incluir el mal funcionamiento del sistema de registro, movimientos generado por el viento, actividades terrestres como perforaciones, construcciones, tráfico de vehículos y actividades humanas y mareas.
En los registros, las frecuencias bajas son debidas al ruido instrumental y se pueden eliminar mediante un filtro pasa-altos con frecuencia de corte de 0.075 Hz, mientras que las frecuencias altas se eliminan con un filtro pasa-bajos con una frecuencia de corte de 10 Hz, ya que las señales sísmicas por lo general se encuentran en estan en ese rango. Ambas funciones se pueden aplicar mediante un filtro pasabanda butterworth de respuesta finita (FIR *Finite Impulsive Response*) de segundo orden con banda de paso de 0.075 a 10 Hz.

4.3.2.5. Normalización

La normalización consiste en el escalado de los datos para que la amplitud de la señal se encuentre en un rango de (-1,1). Se realiza la normalización de las señales para nivelar los valores en el registro sísmico.

4.4. Procesamiento de la señal sísmica

En esta etapa como se muestra en la figura 4.3.2 se realiza la extracción de características. Es fundamental porque aquí se identifican las propiedades representativas de las señales sísmicas en el dominio del tiempo y de la frecuencia [39].

4.4.1. Segmentación

Este proceso consiste en dividir la señal sísmica en partes iguales sin solapamiento [40], cada segmento se guardará en una matriz NumPy. El tamaño de la matriz de características dependerá del tamaño de la ventana y al mismo tiempo del número de segmentos. Un gran número de segmentos aumentará la dimensión de la matriz y viceversa, de ahí que el tamaño de la ventana, y por lo tanto, el número de segmentos se debe de elegir con cuidado [41]. Tomando como referencia a (Ruano.2014) [42], se dividirá la señal en 6 segmentos.

4.4.2. Extracción de características en el dominio del tiempo

En el dominio del tiempo se extraen las siguientes características:

1. **Valor Máximo:** el valor más alto en los datos, dado por:

$$V_{max} = max\left(|x(i)|\right)$$

2. **Valor Mínimo**: el valor más pequeño en los datos, dado por:

$$V_{min} = min\left(|x(i)|\right)$$

3. **Media:** Se obtiene sumando todos los valores y dividiendo por el tamaño de muestra.

$$\overline{X} = \frac{\sum_{i=1}^{N} x_i}{N}$$

4. **Desviación Estándar:** indica que tan dispersos se encuentran en promedio, los datos con respecto a la media aritmética.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} \left(x_i - \overline{X}\right)^2}{N}}$$

5. **Desviación Media Absoluta**: Es una medida de dispersión y se calcula como la media aritmética de los valores absolutos de las desviaciones respecto a la media.

$$D_m = \frac{\sum_{i=1}^{N} \left| x_i - \overline{X} \right|}{N}$$

6. **SMA:** Es el área normalizada de la magnitud de señal dada por:

$$x_{SMA} = \sum_{i=1}^{n} x_i$$

7. **Energia:** Se define como energía de una señal en un intervalo de tiempo para el caso discreto como:

$$E = \sum_{i=1}^{n} |x\,(i)|^2$$

8. **Rango intercuartil IQR:** Es una medida de dispersión similar a la desviación estándar o la varianza, pero mucho más robusta frente a los valores atípicos. El conjunto de datos clasificados se divide en cuatro partes iguales, y se calcula la diferencia entre los percentiles 75 y 25 de los datos.

$$IQR = Q_3 - Q_1$$

9. **Entropía:** se encarga de obtener la incertidumbre de un proceso aleatorio, en este caso de las señales sísmicas. La entropía también se puede interpretar como la cantidad de información promedio, que se necesita para describir dicho proceso.

En total son 9 características por segmento, considerando que son 6 segmentos en total serán 54 características.

4.4.3. Extracción de características en el dominio de la frecuencia

En el dominio de la frecuencia, se extraerán las características:

1. **Densidad Espectral de Potencia PSD.**
2. **Máximo Índice de Densidad Espectral PSD.**
3. **Frecuencia a PSD Max**.

4. **Sesgo Espectral (*Spectral Skewness*).**
5. **Frecuencia Media.**
6. **Kurtosis Espectral.**
7. **Espectro de Potencia a: 0.20 Hz.**
8. **a 1.25 Hz.**
9. **a 2.00 Hz.**
10. **a 3.30 Hz.**
11. **a 8.00 Hz.**

Hacen un total de 11 características por segmento, tratandose de 6 segmentos serán 66 características para cada registro. En la tabla 4.4.1 se muestran las características que se extraerán tanto en el dominio del tiempo como de la frecuencia.

Características en el dominio del tiempo	**Características en el dominio de la frecuencia**
1. $V_{max} = max\left(\left|x(i)\right|\right)$	1. Power Spectral Density PSD at 1.0-3.5 Hz
2. $V_{min} = min\left(\left|x(i)\right|\right)$	2. Max(PSD)
3. Mean $\overline{X} = \frac{\sum_{i=1}^{N} x_i}{N}$	3. Frequency at Max(PSD)
4. Standard Deviation $\sigma = \sqrt{\frac{\sum_{i=1}^{N}\left(x_i - \overline{X}\right)^2}{N}}$	4. Spectral Skewness
5. Absolute Mean Deviation $D_m = \frac{\sum_{i=1}^{N}\left\|x_i - \overline{X}\right\|}{N}$	5. Mean Frequency
6. SMA Signal Magnitude Area $x_{SMA} = \sum_{i=1}^{n} x_i$	6. Spectral Kurtosis
7. Energy $E = \sum_{i=1}^{n} \left\|x\left(i\right)\right\|^2$	7. Power Spectrum at 0.20 Hz
8. Interquartile Range $IQR = Q_3 - Q_1$	8. at 1.25 Hz
9. Entropy	9. at 2.00 Hz
	10. at 3.30 Hz
	11. at 8.00 Hz

Tabla 4.4.1: Características extraídas en el dominio del tiempo y frecuencia.

Capítulo 5

EXPERIMENTACIÓN

El aprendizaje profundo ha demostrado tener muchas capacidades en aplicaciones de reconocimiento de patrones sísmicos. La forma tradicional de hacerlo es proponiendo un modelo basado en el método de prueba y adaptación. Sin embargo se requiere un enfoque mas eficiente a fin de optimizar los hiperparámetros del modelo elegido.

Se propone inicialmente implementar una red neuronal profunda. Una red neuronal profunda *Deep Neural Network* DNN es una red neuronal artificial con múltiples capas entre las capas de entrada y la capa de salida capaz de identificar clases no separables tan facilmente [43].

La arquitectura de la red se definirá a través de un proceso de experimentación. Se requiere una red lo suficientemente grande para capturar la estructura del problema, pero implementar una red con muchas capas ocultas y gran cantidad de neuronas en cada una tiene muchos inconvenientes, como el aumento de la carga computacional, mas tiempo de aprendizaje en la etapa de entrenamiento, además de pérdida en la capacidad de generalización, entre otros problemas.

Los hiperparámetros afectan directamente el proceso de aprendizaje y representan un gran problema de optimización, sin un ajuste adecuado, el algoritmo puede no funcionar adecuadamente, por ello se usará la optimización de hiperparámetros utilizando Talos. La librería Talos permite encontrar los hiperparámetros óptimos en modelos de Keras. Utiliza estrategias de optimización de hiperparámetros de cuadrícula, aleatorias y probabilísticas para maximizar la flexibilidad y la eficiencia [44].

5.1. Procesamiento de la señal sísmica

5.1.1. Obtención de datos y cambio de formato

Los registros se obtienen tomando lde los sismos localizados de la red Sísmica NARS-Baja de sismos mayores o iguales a 3 grados, en el periodo de agosto a diciembre de 2007. Los datos se encuentran en formato SEED. El formato SEED es un formato de datos destinado principalmente para almacenar e intercambiar datos de series de tiempo sismológicas y metadatos. Ya que SEED no está diseñado para el procesamiento, es necesario convertir a formato SAC, que es uno de los principales formatos para tareas de procesamiento de datos. Este no solo incluye datos de forma de onda, incluye además archivos de metadatos y posee herramientas de análisis utilizadas en el estudio de eventos sísmicos.

5.1.2. Preprocesamiento de la señal sísmica

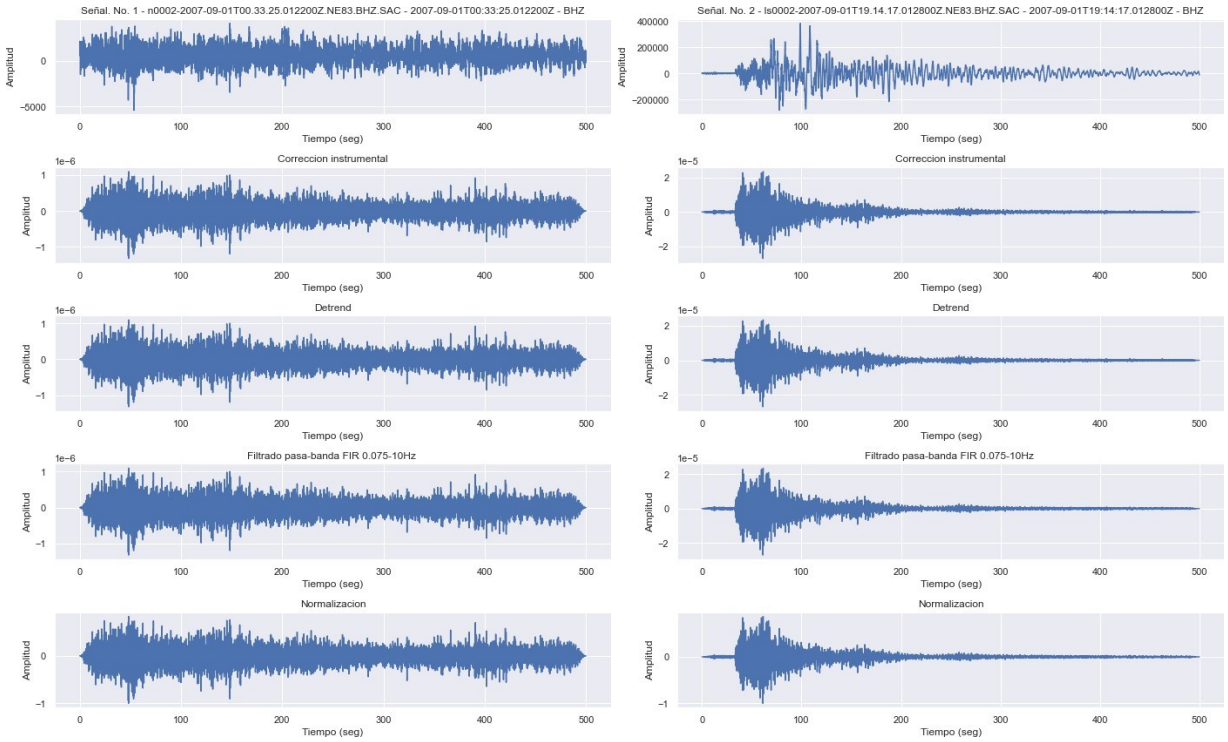
En el preprocesamiento se realizan las tareas de selección de canal (la señal sísmica se genera en 3 canales: 2 canales horizontales, longitudinal y transversal , y un canal vertical, se elige el canal vertical) Después se realiza:

- Corrección instrumental.
- Eliminar tendencia (lineal).
- Filtrado. Filtro Butterworth pasabanda FIR de 2º orden con banda de paso de 0.075 – 10 Hz.
- Normalización.

Estas etapas se muestran en la figura 5.1.1:

5.1.3. Extracción de características

El primer paso en el procesamiento es la segmentación que consiste en dividir el registro sísmico en 6 segmentos no superpuestos de igual tamaño, como se muestra en la figura 5.1.2 y despues realizar la extracción de características en el dominio del tiempo y de la frecuencia (figura 5.1.3) para formar los vectores de características.



(a) Preprocesamiento de ruido sísmico

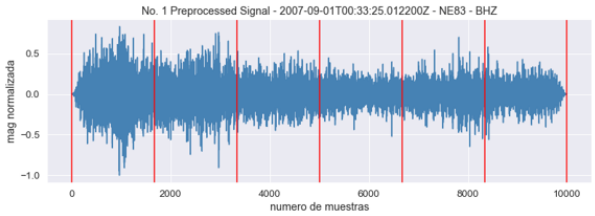
(b) Preprocesamiento de un sismo

Figura 5.1.1: Preprocesamiento de (a) ruido sísmico y (b) un sismo: corrección instrumental, eliminar tendencia (lineal), filtrado y normalización.

Dado que que los registros sísmicos vienen de distancias similares debido a que son réplicas de un sismo importante, se tendrán formas de onda con duraciones similares, es por ello que se puede segmentar en 6 partes iguales. Esto no puede ocurrir en casos generales donde no se conoce la distancia del epicentro (Ortega. 2020).

5.1.4. Conjunto de datos *Dataset*

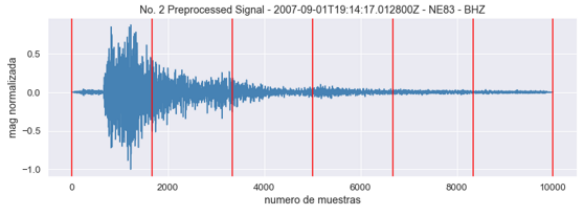
Al final del procesamiento se obtiene el *dataset* (figura 5.1.4) con las características extraídas a cada registro y se almacena en un archivo CSV que contiene 560 registros, 280 etiquetados como "sismo" o clase 1 y 280 como "ruido" o clase 0, cada uno con 120 características.




```
No. 1 Nombre de archivo:  pre-n0002-2007-09-01T00.33.25.012200Z.NE83.BHZ..sac
NR.NE83..BHZ | 2007-09-01T00:33:25.012200Z - 2007-09-01T00:41:44.962200Z | 20.0 Hz, 1000
0 samples
6 segmentos de 83.350 seg, 1666.667 muestras por segmento

Segmentos  6 Trace(s) in Stream:
NR.NE83..BHZ | 2007-09-01T00:33:25.012200Z - 2007-09-01T00:34:48.312200Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T00:34:48.362200Z - 2007-09-01T00:36:11.662200Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T00:36:11.712200Z - 2007-09-01T00:37:35.012200Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T00:37:35.062200Z - 2007-09-01T00:38:58.362200Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T00:38:58.412200Z - 2007-09-01T00:40:21.712200Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T00:40:21.762200Z - 2007-09-01T00:41:44.962200Z | 20.0 Hz, 1665
samples
```

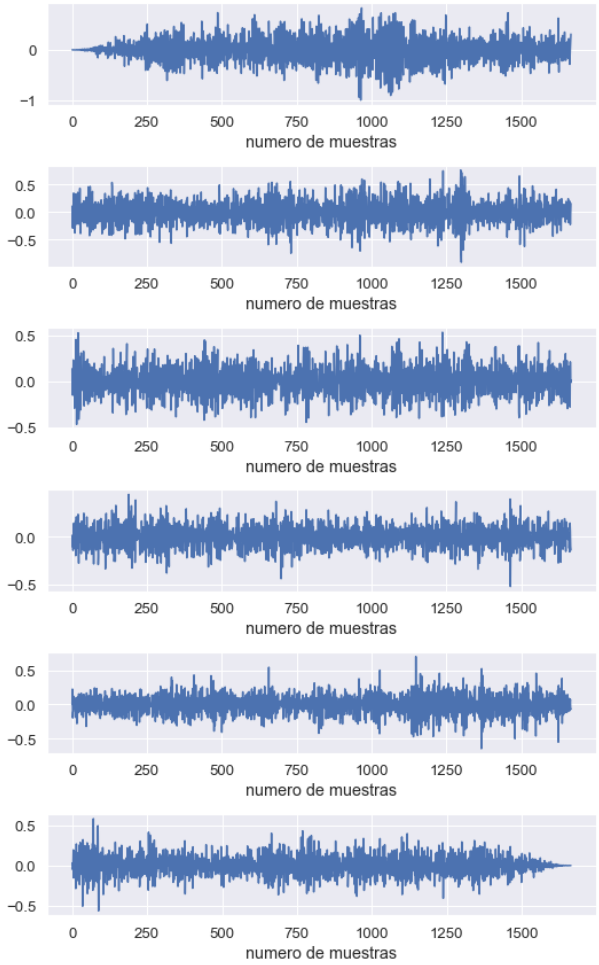
(a) Señal de ruido segmentada



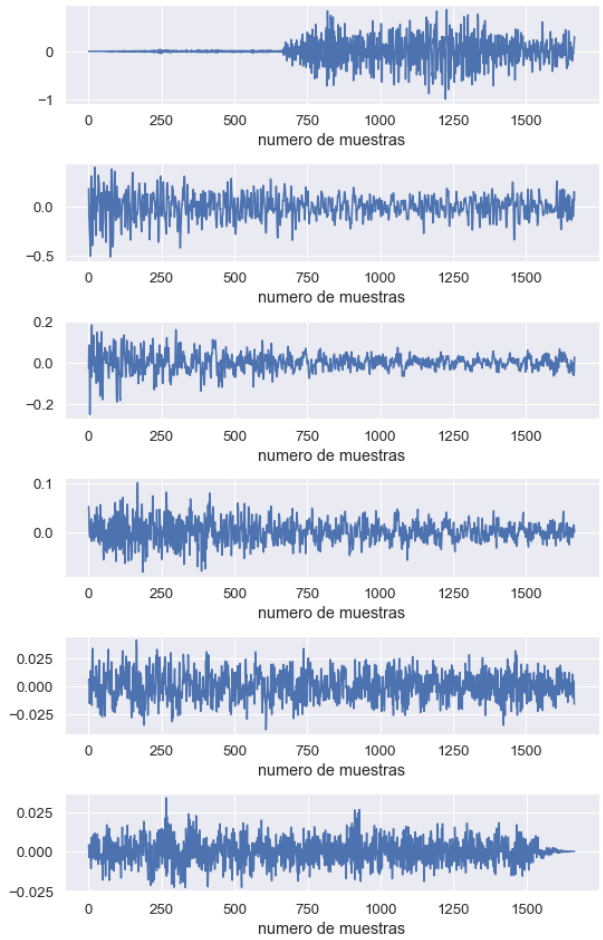

```
No. 2 Nombre de archivo:  pre-ls0002-2007-09-01T19.14.17.012800Z.NE83.BHZ.sac
NR.NE83..BHZ | 2007-09-01T19:14:17.012800Z - 2007-09-01T19:22:36.962800Z | 20.0 Hz, 1000
0 samples
6 segmentos de 83.350 seg, 1666.667 muestras por segmento

Segmentos  6 Trace(s) in Stream:
NR.NE83..BHZ | 2007-09-01T19:14:17.012800Z - 2007-09-01T19:15:40.312800Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T19:15:40.362800Z - 2007-09-01T19:17:03.662800Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T19:17:03.712800Z - 2007-09-01T19:18:27.012800Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T19:18:27.062800Z - 2007-09-01T19:19:50.362800Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T19:19:50.412800Z - 2007-09-01T19:21:13.712800Z | 20.0 Hz, 1667
samples
NR.NE83..BHZ | 2007-09-01T19:21:13.762800Z - 2007-09-01T19:22:36.962800Z | 20.0 Hz, 1665
samples
```

(b) Señal de sismo segmentada



(c) Segmentos obtenidos de ruido sísmico



(d) Segmentos obtenidos de un sismo

Figura 5.1.2: Señal sísmica segmentada en 6 partes para la extracción de características

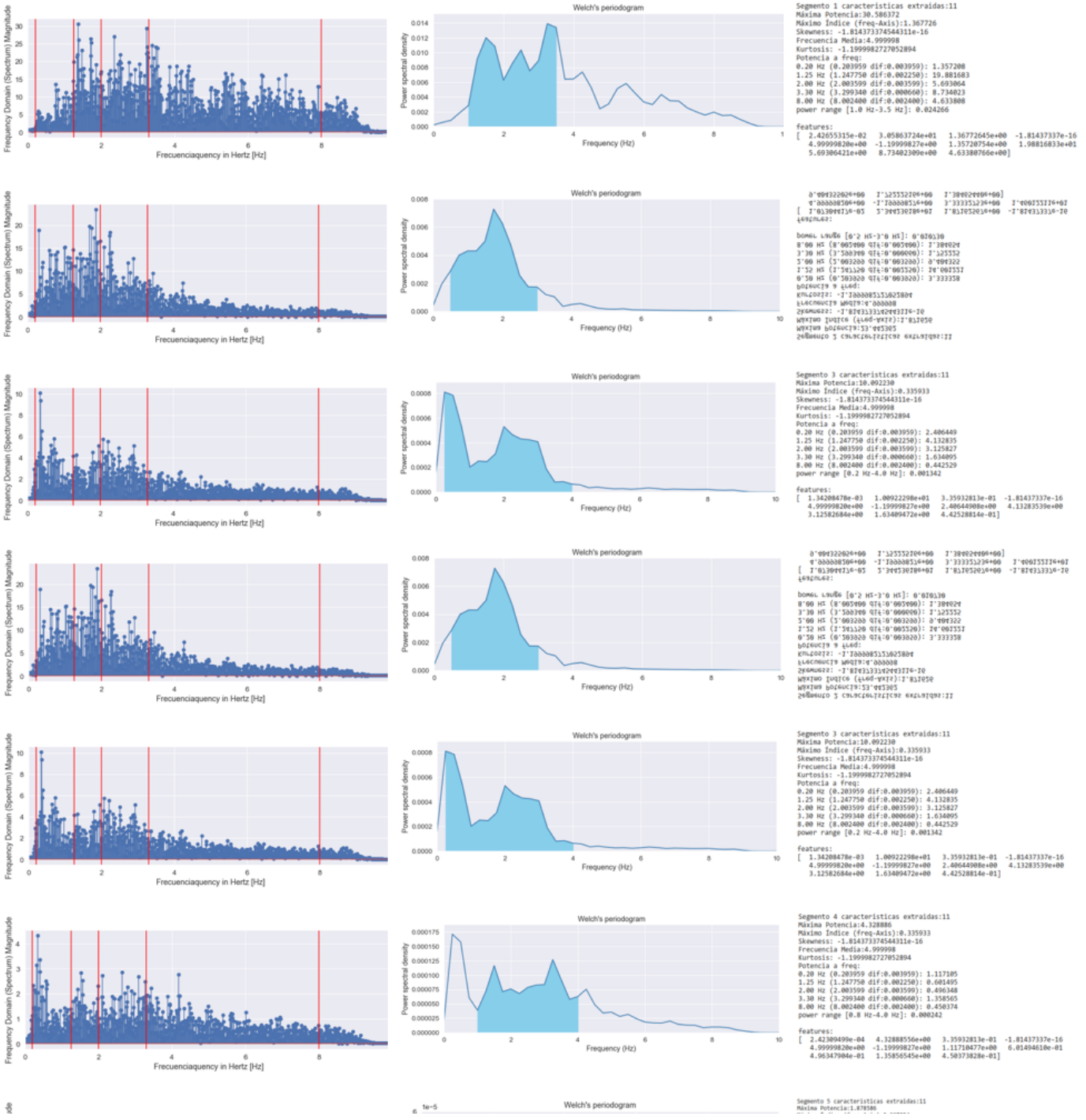


Figura 5.1.3: Características extraídas en el dominio de la frecuencia.

	dmax1	dmin1	dmean1	dstd1	damd1	dsma1	...	1frq6	2frq6	3frq6	4frq6	5frq6
0	0.196177	-0.182893	-0.000032	0.024478	0.016961	-0.053523	...	13.061057	0.129085	0.381834	0.738751	0.102060
1	0.870841	-1.000000	0.000416	0.213352	0.133075	0.693652	...	0.527988	0.203336	0.297131	0.314410	0.371768
2	0.877111	-1.000000	-0.000813	0.179594	0.103380	-1.354459	...	0.196709	0.064095	0.073660	0.038368	0.034759
3	0.841589	-0.589499	-0.000062	0.147390	0.104511	-0.103998	...	0.246751	0.278857	0.903281	1.703511	0.826334
4	1.000000	-0.860030	-0.000345	0.200711	0.122201	-0.575828	...	0.246432	0.088779	0.173094	0.213357	0.252361
...	...	...	...	...	...	...	...	...	...	...	...	...
555	0.698030	-0.726668	0.000017	0.210493	0.161482	0.028704	...	1.280036	20.079727	10.375290	6.698100	2.670943
556	0.751707	-0.756101	-0.000383	0.249670	0.195162	-0.638650	...	22.362734	5.213241	2.209374	17.155539	18.144888
557	0.513080	-0.471594	-0.000248	0.127064	0.098119	-0.413185	...	15.338131	3.580842	4.007958	4.105090	2.234853
558	0.543979	-0.734260	-0.000111	0.187798	0.147014	-0.184624	...	0.910169	0.926057	2.809048	16.510143	7.710167
559	1.000000	-0.945649	-0.000370	0.258895	0.191523	-0.616813	...	5.475055	2.978549	4.066997	2.970926	0.939231

Donde:

dmax	Valor Máximo	peaky	Máxima Potencia max (PSD)
dmin	Valor Mínimo	frqx	Máximo Índice (freq-Axis), frequency at max (PSD)
dmean	Media	skew	Asimetría Skewness, Spectral Skewness
dstd	Desviación Estándar Standard Deviation	frqm	Frecuencia Media, Mean Frequency
damd	Desviación Media Absoluta Absolute Mean Deviation	krt	Kurtosis, Spectral Kurtosis
dsma	SMA: Area Normalizada de la Magnitud, Signal Magnitude Area	1frq	Potencia a freq: 0.20 Hz, Power Spectrum at 0.20 Hz
dsp	Energía de la Señal, Energy	2frq	1.25 Hz Power Spectrum at 1.25 Hz
diqr	IQR Interquartile Range	3frq	2.00 Hz Power Spectrum at 2.00 Hz
dentry	Entropía, Entropy	4frq	3.30 Hz Power Spectrum at 3.30 Hz
pwrrg	Power Range. Densidad Espectro de Potencia PSD en intervalo (1-3.5 Hz) PSD at 1.0-3.5 Hz	5frq	8.00 Hz Power Spectrum at 8.00 Hz

Figura 5.1.4: *Dataset*.

5.2. Red Neuronal Artificial con 120 características

5.2.1. Leer el *dataset*, normalizar y dividir en datos de entrenamiento y prueba

Se carga y lee el *dataset*, y después se normalizan los datos escalandolos dentro del rango de 0 a 1, esto es esencial ya que las variables poseen diferentes unidades. El escalado se lleva a cabo para mejorar la precisión del cómputo numérico posterior y obtener una mejor salida. De salida, se tienen 2 clases categóricas, sismos y ruido, las cuales se codifican. La codificación activa es un proceso para convertir las clases en valores binarios.

El *dataset* contiene 120 características (columnas) y 560 muestras (renglones), se divide de manera aleatoria y se toma el 70 % (392) en muestras de entrenamiento y el 30 % (168) en muestras de prueba.

5.2.2. Definir el diccionario de hiperparámetros

Un diccionario es una estructura de datos que permite almacenar cualquier tipo de valor (enteros, cadenas, listas, funciones) en parejas los cuales contienen un conjunto de claves y valor, lo cual permite identificar a cada valor del diccionario por medio de una clave asociada.

Existen 3 formas de ingresar valores al diccionario:

1. Como rangos escalonados (min, max, pasos)

2. Como valores múltiples [en una lista]

3. Un solo valor [en una lista]

El diccionario se define de la siguiente manera:

```
p = {'first_neuron':            (80,240,8), #valores: 80,100..220,240
        'hidden_layers':        [0,1,2], #  0,1 y 2 capas
        'hidden_neuron':        (80,320,24), #  80,90, ...310,320
        'batch_size':           [16, 32, 64, 128, 256],
        'loss':                         ['binary_crossentropy' ],
        'optimizer':            ['adam','RMSprop',  'adamax'],
        'kernel_initializer':  ['uniform','normal'],
        'epochs':                       [300],
        'last_activation':      ['softmax', 'sigmoid']
        }
```

Cada hiperparametro se presenta en la tabla 5.2.1.

5.2.3. Construir el modelo

Se define el modelo de Keras, haciendo referencia al diccionario de Talos:

```

def model_NN(x_train, y_train, x_test, y_test, params):
        model = Sequential()
        # primera capa
        model.add(Dense(params['first_neuron'], input_dim= x_train.shape[1],
            activation='relu',kernel_initializer= params['kernel_initializer']))
        # genera capas ocultas
        for i in range(params['hidden_layers']):
                print (f"adding layer {i+1}")
                model.add(Dense(params['hidden_neuron'],activation='relu',
                kernel_initializer=params['kernel_initializer']))
        # capa de salida
        model.add(Dense(2, activation=params['last_activation'],
        kernel_initializer=params['kernel_initializer']))
        # se compila el modelo
        model.compile(loss= params['loss'],optimizer=params['optimizer'], metrics
            =['acc'])
        # entrenamiento
        history = model.fit(x_train, y_train,validation_data=[x_test, y_test],
        batch_size=params['batch_size'],epochs=params['epochs'],
            callbacks=[early_stopper(params['epochs'], patience=20)],verbose=0)
        return history, model
```

Parámetro	valor	Característica
first_neuron	(80,240,8)	Número de neuronas de la primera capa oculta. Se le asocia a una lista de valores que van del 80 a 240, en 8 pasos, es decir, los valores que puede tomar son 80, 100, 120,.. 220, 240
first_neuron	[0,1,2]	Número de capas ocultas. Puede tomar los valores de 0, 1 y 2, la arquitectura del modelo puede no tener una capa oculta, o tener 1 o 2 capas
hidden_neuron	(80,320,24)	Número de neuronas ocultas que tienen la(s) capa(s) oculta(s). Valores desde 80 a 320 en 24 pasos. Toma los valores de 80, 90, 100… 300, 310 y 320.
batch_size	[16, 32, 64, 128, 256]	Tamaño de lotes. Toma valores de 16, 32, 64, 128 y 256.
loss	'binary_crossentropy'	Función de perdida. Loss toma la función de perdida entropia cruzada binaria.
optimizer	'adam','RMSprop','adamax'	Función de optimización.
kernel_ initializer	'uniform', 'normal'	La red neuronal debe comenzar con algunos pesos y luego actualizarlos iterativamente para obtener mejores valores. kernel_initializer es la función utilizada para inicializarlos. Se utilizará la distribución uniforme y normal para inicializar los pesos.
Epochs	300	Épocas. Una época se refiere a un ciclo a través del conjunto de datos de entrenamiento completo. Entrenar una red neuronal llevara más de una época en diferentes patrones, esperando una mejor generalización cuando se le da una nueva entrada. La red se entrenara esta cantidad de épocas como máximo.
last_ activation	'softmax', 'sigmoid'	Función de activación. Define la salida de un nodo dada una entrada o un conjunto de entradas. Tiene como objetivo acotar los valores de salida para mantenerlos en ciertos rangos.

Tabla 5.2.1: Definición del diccionario de hiperparámetros para una entrada 120 características.

5.2.4. Configurar parámetros del experimento

Se definen los parametros mostrados en la tabla 5.2.2 para el experimento:

Parámetro	**valor**	**Descripción**
x	x_train	Características de predicción
y	y_train	Variable de resultado de predicción
x_val	x_test	Datos de validación para x
y_val	y_test	Datos de validación para y
model	model_NN	Modelo de Keras
params	p	Diccionario de parámetros
print_params	True	Imprime los hiperparámetros de cada permutación
round_limit	500	Número de permutaciones que se ejecutarán
experiment_name	NNscan500	Crea una carpeta y salva los resultados del experimento
clear_session	True	Borra sesión de backend entre permutaciones
save_weights	True	Salva los pesos del modelo
reduction_metric	'val_acc'	Métrica que se utilizará para la reducción.
minimize_loss	True	Minimizar perdida

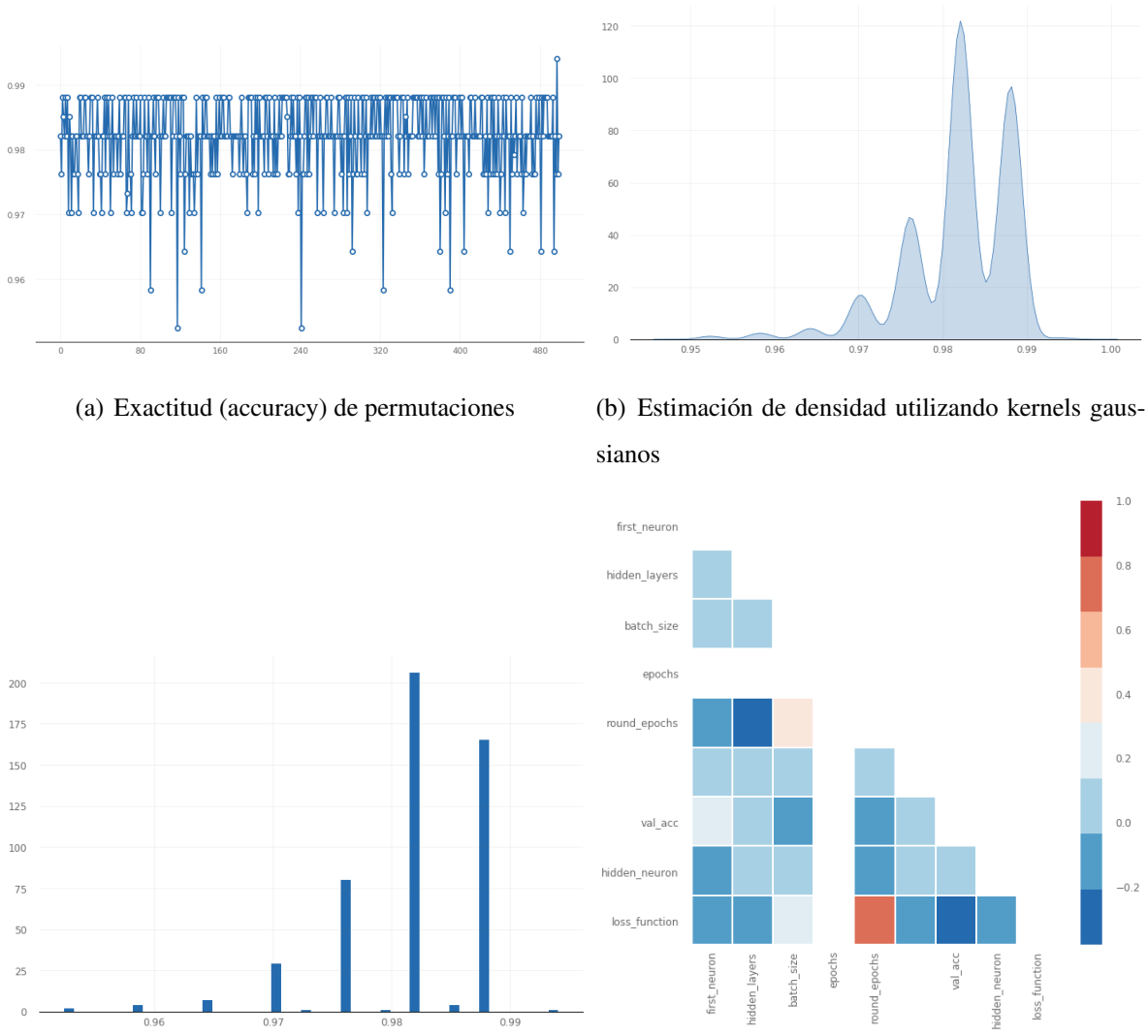
Tabla 5.2.2: Configuración de parámetros del experimento.

5.2.5. Ejecutar experimento de Talos

Se ejecuta el experimento con los parámetros de la tabla 5.2.2. y la configuración de cada capa contenido en el diccionario (tabla 5.2.1.). Para ello se utiliza el comando *scan()* de Talos.

```
start_time = time.time()
t= ta.Scan( x=x_train,
        y=y_train,
        x_val= x_test,
        y_val= y_test,
        model=model_NN,
        params=p,
        print_params= True,
        round_limit= 500,
        experiment_name= experiment,
        clear_session= True,
        save_weights= True,
        reduction_metric= 'val_acc',
        minimize_loss= True
        )
print(t.data)
end_time = time.time()
print("\n--- (%s segundos) ---" % (end_time - start_time))
```

Una vez generadas las permutaciones dentro del espacio de hiperparámetros, es posible acceder a los resultados y mostrar algunas gráficas generadas en el experimento, como las que se muestran en la figura 5.2.1.



(a) Exactitud (accuracy) de permutaciones

(b) Estimación de densidad utilizando kernels gaussianos

(c) Histograma de las permutaciones generadas vs val_acc

(d) Correlación de mapa de calor

Figura 5.2.1: Gráficas generadas en el experimento para la métrica val_acc

5.2.6. Evaluación de los modelos

Los modelos se evalúan contra una validación cruzada de k veces. Idealmente, al menos el 50 % de los datos se mantienen fuera del proceso de entrenamiento en el proceso *scan()* y se exponen al

evaluar los modelos.

Al realizar esta validación cruzada se obtiene:

```
evaluate_object = ta.Evaluate(t)
evaluate_object.evaluate(x_test, y_test, folds=10, metric='val_acc', task='
    multi_label')
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9352226720647774]
```

5.2.7. Implementación del modelo

Se crea un paquete de implementación con el método *Deploy()*. El mejor modelo se elige automáticamente en función de una métrica determinada ('*val_acc*' por defecto). El paquete *Deploy* es un archivo .zip que consta de: detalles de *scan()*, los pesos, la estructura del mejor modelo de red neuronal, etc. que pueden usarse para realizar predicciones (tabla 5.2.7).

Archivo	**Descripción**
NNscan500_val_acc_deploy_details.txt	Resumen de experimento de talos
NNscan500_val_acc_deploy_model.h5	Mejor modelo en las permutaciones de acuerdo a la métrica especificada (acc, val_acc o val_loss)
NNscan500_val_acc_deploy_model.json	Estructura del mejor modelo de la Red Neuronal de acuerdo a la métrica
NNscan500_val_acc_deploy_params.npy	
NNscan500_val_acc_deploy_results.csv	Resultados de las permutaciones generadas en el experimento en formato csv
NNscan500_val_acc_deploy_x.csv	Etiquetas de las entradas x
NNscan500_val_acc_deploy_y.csv	Etiquetas de salida y
README.txt	Instrucciones y formato para el uso del objeto por medio de Restore()

Tabla 5.2.3: Activos del objeto scan() asociados al experimento.

5.2.8. Restauración del modelo

Se restaura el paquete generado el cual consta de los activos del objeto. La libreria Keras brinda la posibilidad de guardar la arquitectura y los pesos de un modelo de Red Neuronal. Los pesos se

guardan en formato HDF5 y la estructura del modelo utilizando el formato json. `JSON` es un formato para intercambio basado en texto para serializar datos.

5.2.9. Cargar el modelo y analizar resultados

El modelo obtenido de acuerdo a la métrica *val_acc* se muestra en la tabla 5.2.4:

first_neuron	100
hidden_layers	2
hidden_neuron	270
batch_size	16
loss_function	binary_crossentropy
acc	1.0
loss	2.63042e-05
val_acc	0.994048
val_loss	0.0869167

Tabla 5.2.4: Modelo de red neuronal con 120 características.

5.2.10. Informe de las principales tasas y métricas de clasificación

El reporte de las principales tasas y métricas de clasificacion obtenidas con el modelo se muestran en las tablas 5.2.5 y 5.2.6.

Accuracy:	99.40 % (0.994048)
Precision:	100.00 % (1.000000)
Recall:	98.88 % (0.988764)

Tabla 5.2.5: Métricas de clasificación.

classification_report

	precision	recall	f1-score	support
0	1.00	0.99	0.99	79
1	0.99	1.00	0.99	89
accuracy			0.99	168
macro avg	0.99	0.99	0.99	168
weighted avg	0.99	0.99	0.99	168

Tabla 5.2.6: Reporte de clasificación.

Errores de clasificación

De acuerdo a la tabla 5.2.6, se comete un error de clasificación, 1 falso positivo en la muestra de 114 como se presenta en la tabla 5.2.7. La forma de onda de registro se muestra en la figura 5.2.2.

#	index	y_real	y_pred	clasificación
1	114	1	0	falso positivo x
114	NE83	2007-09-01T20:20:21.012900Z		

Total= 1 falso(s) positivo(s) 0 falso(s) negativo(s)

Tabla 5.2.7: Errores de clasificación.

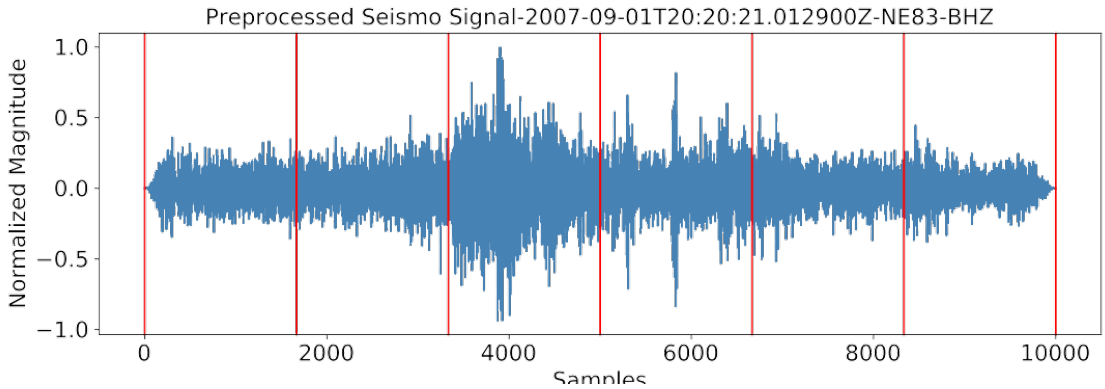


Figura 5.2.2: Forma de onda del registro mal clasificado con la red neuronal de 120 caracteristicas.

Matriz de confusión

La matriz de confusión se muestra en la tabla 5.2.8 y en la figura 5.2.3

Pérdida logarítmica

Best Log-Loss (validation): 0.05607054618719433

La pérdida logarítmica (*log-loss Logarithmic Loss*) es una función de pérdida de clasificación. Minimizar *log-loss* es equivalente a maximizar la precisión del clasificador, por lo que debe estar cerca de cero para un buen algoritmo predictivo.

	Predicted Seismo	Predicted Noise
Real Seismo	88	1
Real Noise	0	79

True Positives TP:	88
False Positives FP:	1
True Negatives TN:	0
False Negatives FN:	79

Tabla 5.2.8: Matriz de confusión.

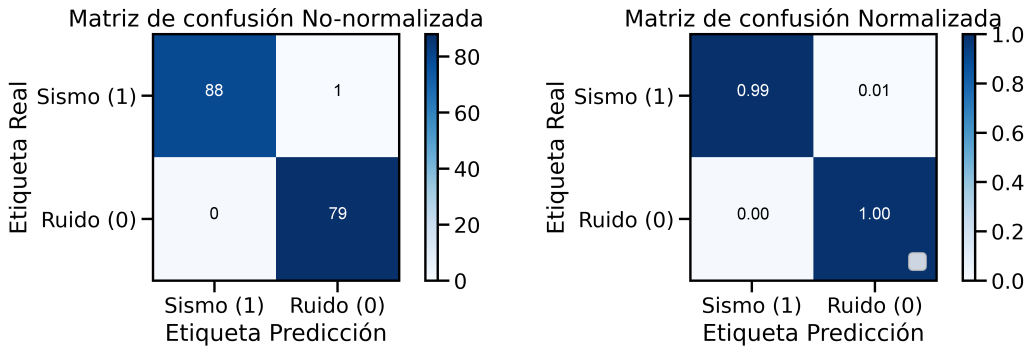


Figura 5.2.3: Matriz de confusión.

Coeficiente de determinación R^2

Coeficiente de determinación (R^2): 0.976105817095719

Un valor cercano a 1 indica un modelo con un error cercano a cero, y un valor cercano a cero indica un modelo muy cercano a la línea de base.

Curva ROC / AUC

AUC: 0.997867 AUC-*Area Under the ROC Curve.*

Porcentaje de AUC Área bajo la curva ROC.

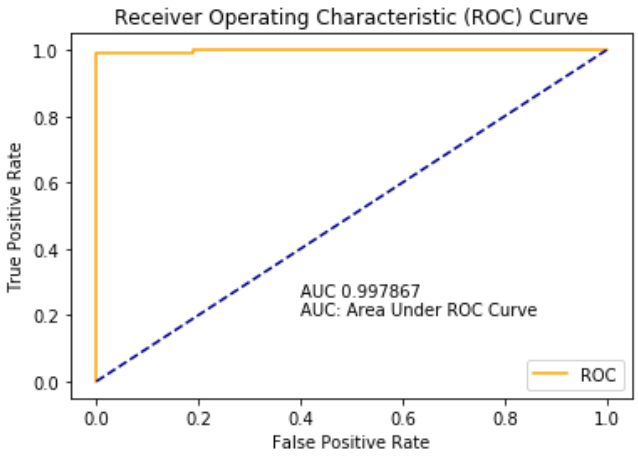


Figura 5.2.4: Área bajo la curva ROC.

5.2.11. Arquitectura y rendimiento del modelo obtenido con 120 características

En la tabla 5.2.9 y figura 5.2.5 se presentan la distribución de capas y neuronas que debera tener la red neuronal con 120 caracteristicas, y en la tabla 5.2.10 la arquitectura y rendimiento del modelo.

```
Model: "sequential_1
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               12100
_________________________________________________________________
dense_2 (Dense)              (None, 270)               27270
_________________________________________________________________
dense_3 (Dense)              (None, 270)               73170
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 542
=================================================================
Total params: 113,082
Trainable params: 113,082
Non-trainable params: 0
```

Tabla 5.2.9: Capas y número de neuronas de la red neuronal con 120 características.

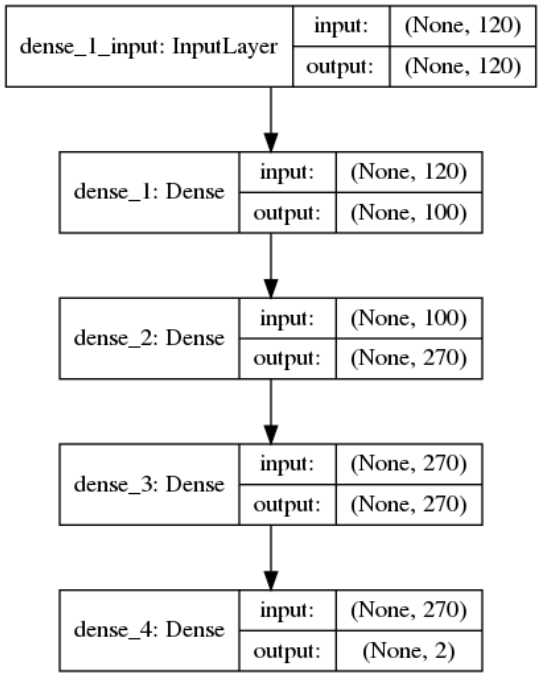
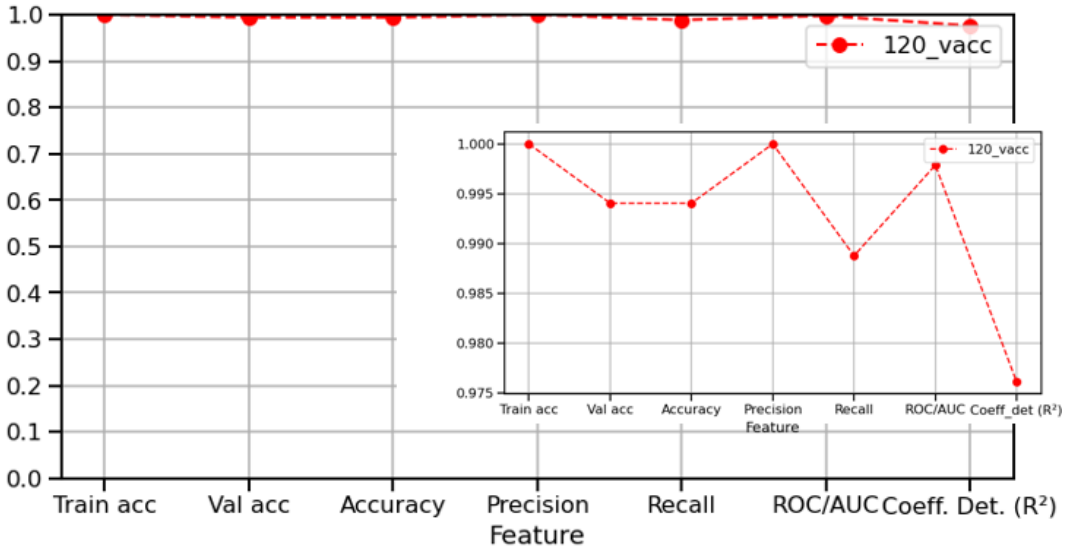


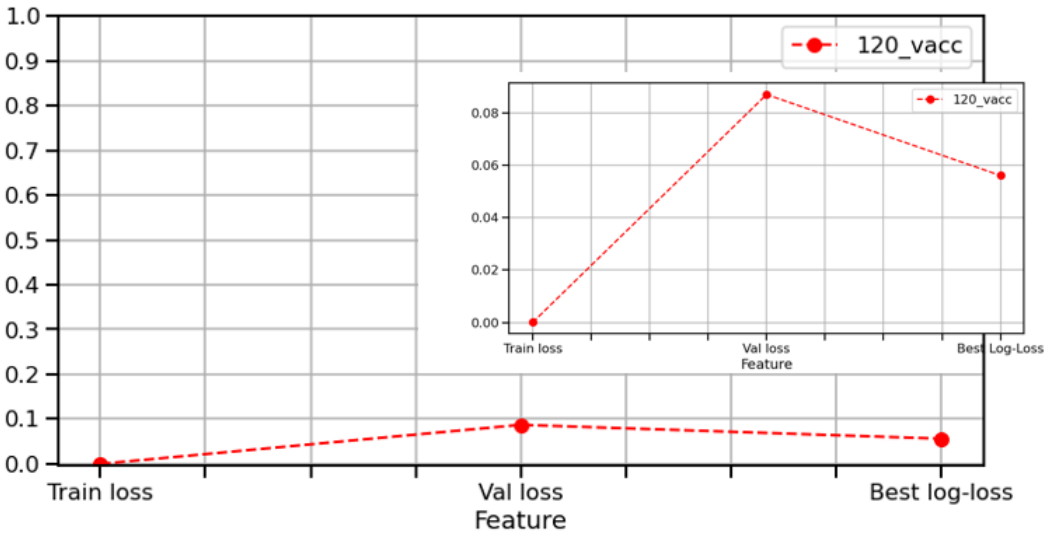
Figura 5.2.5: Distribución de capas de la red neuronal de 120 características.

Feature	**120_vacc**
archivo	NNscan500_val_acc
first_neuron	100
hidden_layers	2
hidden_neuron	270
batch_size	16
loss_function	binary_crossentropy
optimizer	Adam
kernel_initializer	normal
last_activation	softmax
train acc	1
train loss	2.52E-05
val acc	0.994048
val loss	0.086986
accuracy	0.994048
precision	1
recall	0.988764
ROC/AUC Area Under the Curve	0.997867
Best Log-Loss	0.0560705
coeficiente de determinación (R^2)	0.976106
False Positive FP	1
False Negative FN	0

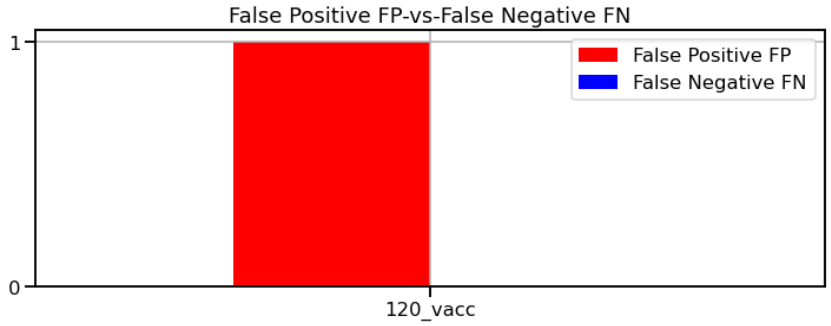
Tabla 5.2.10: Arquitectura y rendimiento del modelo de Red Neuronal Artificial final con 120 características.



(a) Rendimiento del modelo



(b) Pérdidas del modelo



(c) Errores de clasificación

Figura 5.2.6: Rendimiento del modelo de red neuronal artificial con 120 características

Representación gráfica de la Red Neuronal Artificial con 120 características

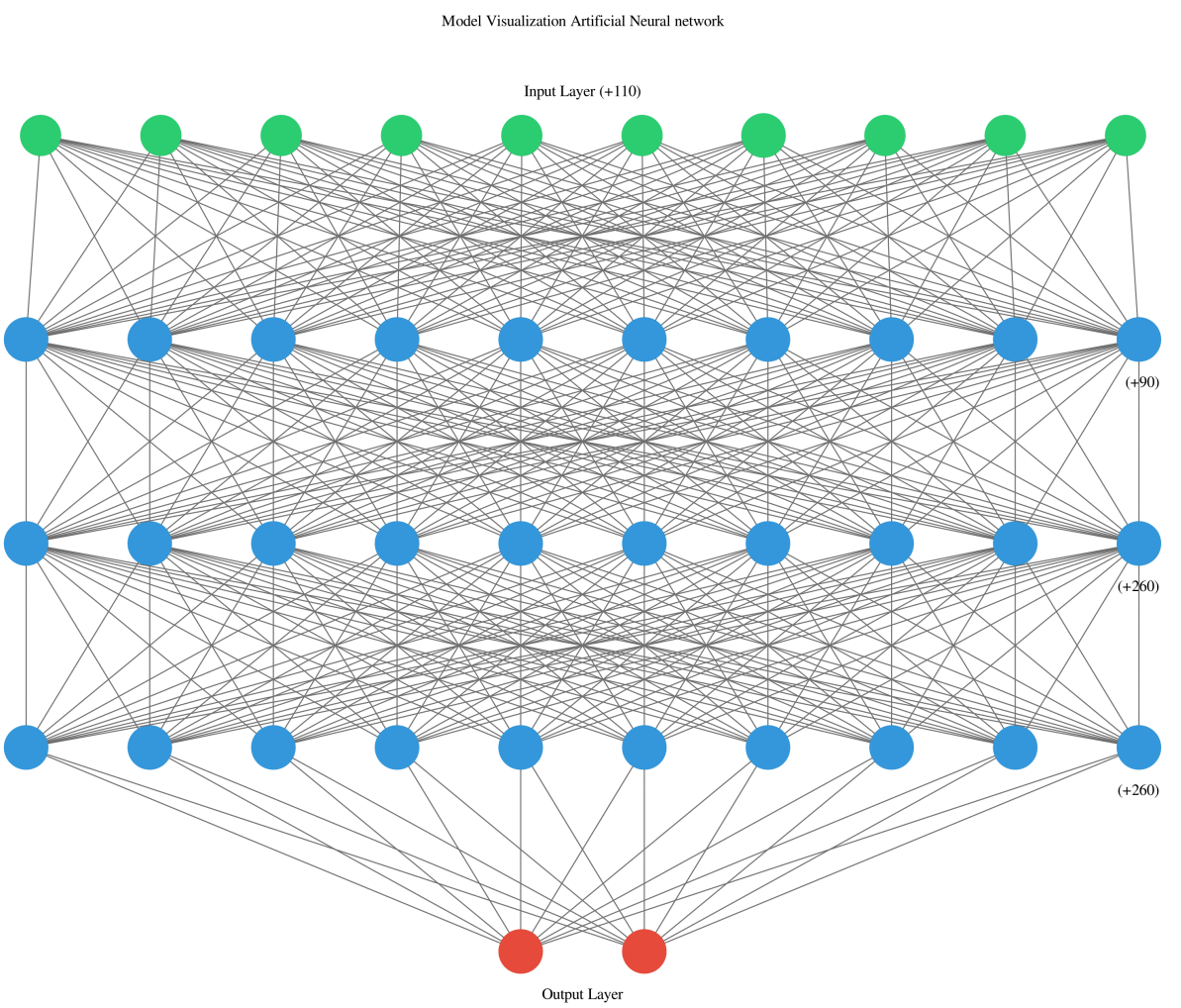


Figura 5.2.7: Representación gráfica de la red neuronal artificial con 120 características.

5.3. Red Neuronal Artificial aplicando reducción de dimensionalidad

5.3.1. Aplicar reducción de dimensionalidad al conjunto de datos de entrada

Se realizará un análisis de la dimensionalidad del conjunto de datos de entrada y se diseñará un nuevo modelo de red neuronal con el *dataset* resultante. Las etapas que se realizarán son:

1. **Preprocesamiento de los datos**

   Incluye las siguientes tareas:

   a) Cargar el *dataset*.

   b) Normalizar los datos.

   c) Realizar análisis de correlación entre las variables de entrada.

   d) Aplicar análisis de varianza explicada entre las variables.

2. **Reducción de dimensionalidad.**

   Se aplicarán técnicas de reducción de dimensionalidad.

   a) Selección de características.

   Se tomarán las características que capturan la mayor parte de la varianza en los datos. Dentro de las técnicas de selección de características, se aplicará:

   - Relación de valores faltantes.
   - Filtro de baja varianza.
   - Filtro de alta correlación.
   - Importancia de las características.

   b) Extracción de características.

   Crea un conjunto más pequeño de nuevas variables que resultan de la combinación de las variables originales. Para la extracción de características se aplicará:

   - Análisis de Componentes Principales PCA.

5.3.2. Preprocesamiento de los datos

5.3.2.1. Cargar el *dataset*

```
def load_data(filename):
        data = pd.read_csv(filename)
        return data

filename ='dataset_hd.csv'
data= load_data(filename)
data
```

	Clase	Estacion	Fecha	Canal	dmax1	dmin1	dmean1	...	2frq6	3frq6	4frq6	5frq6
0	1	NE74	1/9/2007 T19:14:17.009300Z	BHZ	0.196177	-0.182893	-0.000032	...	0.129085	0.381834	0.738751	0.10206
1	1	NE83	1/9/2007 T19:14:17.012800Z	BHZ	0.870841	-1	0.000416	...	0.203336	0.297131	0.31441	0.371768
2	1	NE82	1/9/2007 T19:14:17.025300Z	BHZ	0.877111	-1	-0.000813	...	0.064095	0.07366	0.038368	0.034759
3	1	NE83	1/9/2007 T19:38:15.012800Z	BHZ	0.841589	-0.589499	-0.000062	...	0.278857	0.903281	1.703511	0.826334
4	1	NE82	1/9/2007 T19:38:15.025300Z	BHZ	1	-0.86003	-0.000345	...	0.088779	0.173094	0.213357	0.252361
...	...	...	...	...	...	...	...	...	...	...	...	...
555	0	NE74	23/9/2007 T12:50:25.009300Z	BHZ	0.69803	-0.726668	0.000017	...	20.079727	10.37529	6.6981	2.670943
556	0	NE82	23/9/2007 T12:50:25.016300Z	BHZ	0.751707	-0.756101	-0.000383	...	5.213241	2.209374	17.15553	18.144888
557	0	NE82	24/9/2007 T23:46:46.015700Z	BHZ	0.51308	-0.471594	-0.000248	...	3.580842	4.007958	4.10509	2.234853
558	0	NE83	2/9/2007 4T23:46:46.031800Z	BHZ	0.543979	-0.73426	-0.000111	...	0.926057	2.809048	16.51014	7.710167
559	0	NE82	26/9/2007 T06:03:43.015200Z	BHZ	1	-0.945649	-0.00037	...	2.978549	4.066997	2.970926	0.939231

560 rows × 124 columns

Tabla 5.3.1: Dataset con 120 características, 560 registros y 2 clases.

5.3.2.2. Normalizar los datos

Las muestras se normalizarán con la función de scikit-learn *StandardScaler*. La estandarización consiste en centrar las columnas de características con respecto a la media 0 con desviación estándar 1 de manera que las columnas de características tengan una distribución normal estándar con media cero y varianza 1. De esta forma, se mantiene información útil sobre valores atípicos haciendo los algoritmos menos sensibles a ellos (tabla 5.3.2).

$$z = \frac{(x - \mu)}{\sigma}$$

Donde:

μ es la media

σ es la desviación estándar

```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
x_std  = std_scaler.fit_transform(x)
dfx_std = pd.DataFrame(data = x_std , columns = x.columns)
dfx_std
```

	dmax1	dmin1	dmean1	dstd1	damd1	dsma1	...	1frq6	2frq6	3frq6	4frq6	5frq6
0	-2.5304	2.656896	-0.11532	-2.659169	-2.294167	-0.115351	...	3.232961	-0.574969	-0.673714	-0.655192	-0.742822
1	0.364267	-0.883896	1.631693	0.790882	0.35318	1.631689	...	-0.432146	-0.561328	-0.696053	-0.754808	-0.676377
2	0.391172	-0.883896	-3.157194	0.174245	-0.323852	-3.157195	...	-0.529023	-0.586909	-0.75499	-0.81961	-0.759403
3	0.238761	0.894941	-0.233423	-0.41401	-0.298062	-0.233372	...	-0.514389	-0.547453	-0.536191	-0.428711	-0.56439
4	0.918431	-0.277361	-1.336602	0.559981	0.105261	-1.336603	...	-0.514482	-0.582374	-0.728765	-0.77853	-0.705794
...	...	...	...	...	...	...	...	...	...	...	...	...
555	-0.377184	0.300542	0.07684	0.738653	1.000866	0.076911	...	-0.21222	3.090321	1.961889	0.74379	-0.109951
556	-0.14688	0.173	-1.48349	1.454277	1.768737	-1.483494	...	5.953095	0.359082	-0.191731	3.198716	3.702219
557	-1.170718	1.405862	-0.956312	-0.785295	-0.443796	-0.956313	...	3.898857	0.059181	0.282614	0.13507	-0.217386
558	-1.038142	0.267645	-0.421888	0.324108	0.670982	-0.421892	...	-0.320383	-0.428551	-0.033578	3.047207	1.131515
559	0.918431	-0.648373	-1.432433	1.622779	1.685779	-1.432435	...	1.014549	-0.051471	0.298185	-0.13118	-0.536576

560 rows × 120 columns

Tabla 5.3.2: *Dataset* normalizado con la función *StandardScaler*.

5.3.2.3. Análisis de la correlación entre variables

Se analizan inicialmente las muestras, y la correlación que existe entre ellas, para ello se obtiene la matriz de correlación con mapa de calor.

```
import seaborn as sns

dfx_std_corr=dfx_std.corr()

# Genera una mascara para el triangulo superior
mask = np.triu(np.ones_like(dfx_std_corr, dtype=np.bool))

f, ax = plt.subplots(figsize=(20,20))

# Genera mapa de calor personalizado
cmap = sns.diverging_palette(10, 220, sep=80, n=7)

# grafica mapa de calor con la mascara
sns.heatmap(dfx_std_corr, mask=mask, cmap=cmap, vmax=.5, center=0,
square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

En la figura 5.3.1 se observa que existe alta correlación con algunas características.

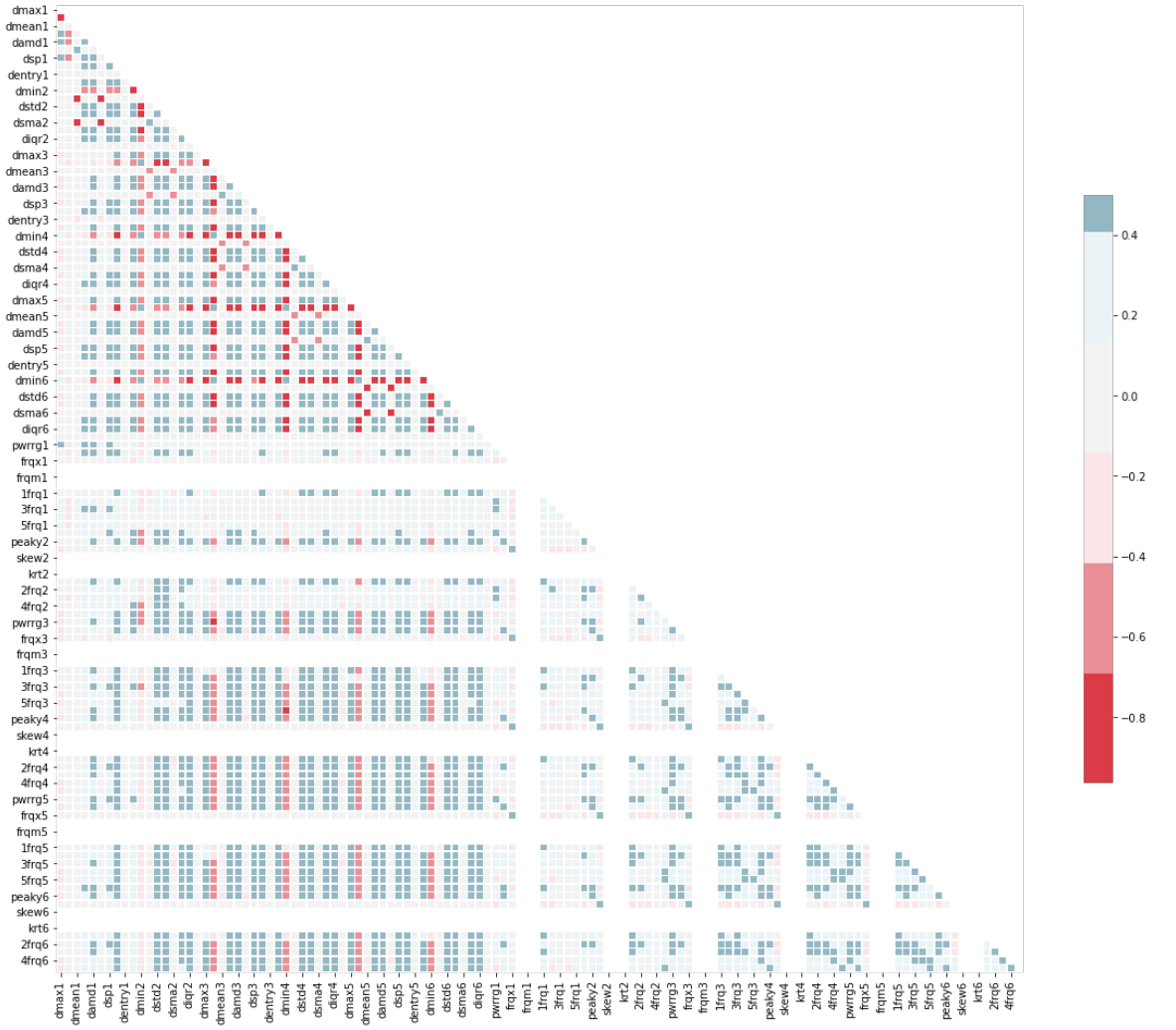


Figura 5.3.1: Matriz de correlación entre las 120 características de entrada.

5.3.2.4. Análisis de la varianza explicada

En la tabla 5.3.3 se analiza la aportación de varianza de cada característica. Se observa que con 91 componentes se explica el 100 % y a partir de ahi, no existe mas aportacion de varianza al aumentar el número de características, por lo que es posible aplicar técnicas de reducción de dimensionalidad para evitar la redundancia y reducir la carga y complejidad computacional.

5.3.3. Técnicas de reducción de dimensionalidad

Despues de determinar que con 91 características se explica el 100 % de la varianza, se aplicará reducción de dimensionalidad mediante selección y extracción de características.

Comp.	varianza expl	acum	%acum	Comp.	varianza expl	acum	%acum
1	0.386536	0.386536	(38.653601)	61	0.001066	0.992133	(99.213340)
2	0.084557	0.471093	(47.109299)	62	0.001034	0.993168	(99.316757)
3	0.048609	0.519702	(51.970249)	63	0.000919	0.994087	(99.408666)
4	0.046963	0.566665	(56.666517)	64	0.000695	0.994782	(99.478178)
5	0.040638	0.607303	(60.730295)	65	0.000642	0.995424	(99.542359)
6	0.031885	0.639188	(63.918784)	66	0.000541	0.995965	(99.596463)
7	0.028755	0.667943	(66.794276)	67	0.000497	0.996461	(99.646136)
8	0.025957	0.693900	(69.389953)	68	0.000464	0.996925	(99.692519)
9	0.022304	0.716203	(71.620315)	69	0.000396	0.997321	(99.732081)
10	0.015930	0.732134	(73.213351)	70	0.000334	0.997655	(99.765492)
11	0.014197	0.746331	(74.633057)	71	0.000320	0.997975	(99.797499)
12	0.013266	0.759596	(75.959620)	72	0.000280	0.998255	(99.825483)
13	0.012820	0.772416	(77.241616)	73	0.000277	0.998532	(99.853191)
14	0.011721	0.784137	(78.413703)	74	0.000254	0.998786	(99.878561)
15	0.010400	0.794537	(79.453715)	75	0.000227	0.999013	(99.901308)
16	0.010255	0.804792	(80.479190)	76	0.000214	0.999227	(99.922718)
17	0.010003	0.814795	(81.479538)	77	0.000186	0.999413	(99.941320)
18	0.009658	0.824453	(82.445313)	78	0.000149	0.999563	(99.956256)
19	0.009071	0.833524	(83.352395)	79	0.000122	0.999685	(99.968463)
20	0.008702	0.842226	(84.222565)	80	0.000120	0.999804	(99.980445)
21	0.008349	0.850575	(85.057468)	81	0.000069	0.999873	(99.987300)
22	0.008081	0.858656	(85.865600)	82	0.000056	0.999929	(99.992902)
23	0.007252	0.865908	(86.590843)	83	0.000029	0.999958	(99.995847)
24	0.006944	0.872852	(87.285247)	84	0.000019	0.999977	(99.997736)
25	0.006446	0.879298	(87.929831)	85	0.000007	0.999984	(99.998449)
26	0.006178	0.885476	(88.547594)	86	0.000006	0.999990	(99.999024)
27	0.005926	0.891402	(89.140157)	87	0.000004	0.999994	(99.999380)
28	0.005775	0.897177	(89.717684)	88	0.000002	0.999996	(99.999624)
29	0.005375	0.902552	(90.255164)	89	0.000002	0.999998	(99.999788)
30	0.004951	0.907503	(90.750253)	90	0.000001	0.999999	(99.999926)
31	0.004810	0.912312	(91.231208)	91	0.000001	1.000000	(100.000000)
32	0.004654	0.916966	(91.696640)	92	0	1.000000	(100.000000)
33	0.004503	0.921469	(92.146948)	93	0	1.000000	(100.000000)
34	0.004240	0.925710	(92.570994)	94	0	1.000000	(100.000000)
35	0.003912	0.929622	(92.962231)	95	0	1.000000	(100.000000)
36	0.003753	0.933376	(93.337579)	96	0	1.000000	(100.000000)
37	0.003651	0.937026	(93.702633)	97	0	1.000000	(100.000000)
38	0.003506	0.940532	(94.053220)	98	0	1.000000	(100.000000)
39	0.003419	0.943951	(94.395096)	99	0	1.000000	(100.000000)
40	0.003347	0.947298	(94.729824)	100	0	1.000000	(100.000000)
41	0.003103	0.950401	(95.040103)	101	0	1.000000	(100.000000)
42	0.002978	0.953379	(95.337855)	102	0	1.000000	(100.000000)
43	0.002828	0.956207	(95.620672)	103	0	1.000000	(100.000000)
44	0.002779	0.958986	(95.898617)	104	0	1.000000	(100.000000)
45	0.002679	0.961665	(96.166502)	105	0	1.000000	(100.000000)
46	0.002625	0.964290	(96.428984)	106	0	1.000000	(100.000000)
47	0.002465	0.966755	(96.675487)	107	0	1.000000	(100.000000)
48	0.002377	0.969132	(96.913155)	108	0	1.000000	(100.000000)
49	0.002334	0.971466	(97.146552)	109	0	1.000000	(100.000000)
50	0.002309	0.973774	(97.377424)	110	0	1.000000	(100.000000)
51	0.002156	0.975930	(97.592994)	111	0	1.000000	(100.000000)
52	0.002039	0.977968	(97.796849)	112	0	1.000000	(100.000000)
53	0.001933	0.979902	(97.990193)	113	0	1.000000	(100.000000)
54	0.001870	0.981771	(98.177147)	114	0	1.000000	(100.000000)
55	0.001774	0.983545	(98.354532)	115	0	1.000000	(100.000000)
56	0.001696	0.985242	(98.524159)	116	0	1.000000	(100.000000)
57	0.001638	0.986879	(98.687911)	117	0	1.000000	(100.000000)
58	0.001504	0.988383	(98.838277)	118	0	1.000000	(100.000000)
59	0.001386	0.989769	(98.976874)	119	0	1.000000	(100.000000)
60	0.001298	0.991067	(99.106694)	120	0	1.000000	(100.000000)

* A partir de 91 componentes se alcanza el 100 % y ya no existe aportación con mas variables.

Tabla 5.3.3: Razón de varianza y porcentaje explicado en las variables de entrada.

5.3.3.1. Selección de características

Buscan mantener las variables más relevantes del conjunto de datos original.

5.3.3.1.1. Relación de valores faltantes

Se exploran los datos buscando si existen valores faltantes y y si estos superan cierto umbral se descarta la variable.

```
# comprobar el porcentaje de valores faltantes en cada variable
x.isnull().sum()/len(x)*100
```

dmax1	0.0
dmin1	0.0
dmean1	0.0
dstd1	0.0
damd1	0.0
...	...
1frq6	0.0
2frq6	0.0
3frq6	0.0
4frq6	0.0
5frq6	0.0

Length: 120 dtype: float64

Tabla 5.3.4: Relación de valores faltantes.

En la tabla 5.3.4 se observa que no se presentan valores faltantes.

5.3.3.1.2. Filtro de baja varianza

Esta técnica calcula la varianza de cada variable a fin de eliminar las variables que tienen una baja varianza en comparación con las demás variables del conjunto de datos. Las variables con baja varianza no afectan de manera significativa a la variable objetivo, por lo que pueden descartarse.

Para ello primeramente se obtiene la varianza,como se muestra en la tabla 5.3.5.

```
# calcular la varianza de todas las variables
x.var()
```

Despues se ordenan en orden ascendente (tabla 5.3.6) y se eliminan las que presentan una baja o nula varianza ya que no aportan o muy poco a la solución.

dmax1	5.441933e-02
dmin1	5.334982e-02
dmean1	6.593908e-08
dstd1	3.002410e-03
damd1	1.927183e-03
...	...
1frq6	1.171434e+01
2frq6	2.968065e+01
3frq6	1.440282e+01
4frq6	1.817816e+01
5frq6	1.650570e+01
Length: 120 dtype: float64	

Tabla 5.3.5: Varianza de los datos de entrada.

```
var_data = x.var().sort_values()
df_var_data = pd.DataFrame(data=var_data,columns = ['var'])
#mostrar las primeras 30 variables
df_var_data.head(30)
```

variable	varianza
skew6	0.000000e0
skew4	1.646205e-60
skew1	1.646205e-60
skew2	1.646205e-60
skew3	1.646205e-60
...	...
dmean1	6.593908e-08
dmean3	8.533366e-08
dmean4	9.152352e-08
dmean2	1.052454e-07
dmean5	1.141679e-07

Tabla 5.3.6: Variables con menor varianza en los datos en orden ascendente.

Estas 30 características tienen poca varianza y se pueden eliminar del conjunto de datos original. de esta manera se logra reducir de 120 a 90 características de entrada mostrado en la tabla 5.3.7.

5.3.3.1.3. Filtro de alta correlación

La alta correlación entre dos variables significa que presentan una tendencia similar y es probable que contengan información similar. Esto puede reducir drásticamente el rendimiento de algunos modelos. Para aplicar un filtro de alta correlación, se calcula la correlación entre variables independientes, si el coeficiente de correlación supera un valor umbral, se puede descartar una de las variables.
Para ello, se calcula la correlación del conjunto de datos, como se muestra en la tabla 5.3.8 y se grafica la matriz de correlación como mapa de calor (figura 5.3.2).

no.	dmax1	dmin1	dstd1	damd1	dsma1	dsp1	...	1frq6	2frq6	3frq6	4frq6	5frq6
0	-2.530400	2.656896	-2.659169	-2.294167	-0.115351	-1.756876	...	3.232961	-0.574969	-0.673714	-0.655192	-0.742822
1	0.364267	-0.883896	0.790882	0.353180	1.631689	0.763159	...	-0.432146	-0.561328	-0.696053	-0.754808	-0.676377
2	0.391172	-0.883896	0.174245	-0.323852	-3.157195	0.019011	...	-0.529023	-0.586909	-0.754990	-0.819610	-0.759403
3	0.238761	0.894941	-0.414010	-0.298062	-0.233372	-0.571777	...	-0.514389	-0.547453	-0.536191	-0.428711	-0.564390
...	...	...	...	...	...	...	...	...	...	...	...	...
556	-0.146880	0.173000	1.454277	1.768737	-1.483494	1.706538	...	5.953095	0.359082	-0.191731	3.198716	3.702219
557	-1.170718	1.405862	-0.785295	-0.443796	-0.956313	-0.884734	...	3.898857	0.059181	0.282614	0.135070	-0.217386
558	-1.038142	0.267645	0.324108	0.670982	-0.421892	0.188073	...	-0.320383	-0.428551	-0.033578	3.047207	1.131515
559	0.918431	-0.648373	1.622779	1.685779	-1.432435	1.969725	...	1.014549	-0.051471	0.298185	-0.131180	-0.536576

560 rows × 90 columns

Tabla 5.3.7: Dataset resultante con 90 variables después de aplicar filtro de baja varianza.

```
import seaborn as sns

dfx_corr=dfx_reduced.corr()

mask = np.triu(np.ones_like(dfx_corr, dtype=np.bool))
f, ax = plt.subplots(figsize=(12, 12))

# Genera mapa de colores personalizado
cmap = sns.diverging_palette(10, 220, sep=50, n=10)
sns.heatmap(dfx_corr, mask=mask, cmap=cmap, vmax=.5, center=0, square=True,
    linewidths=.5, cbar_kws={"shrink": .5})
```

	dmax1	dmin1	dstd1	damd1	dsma1	dsp1	...	1frq6	2frq6	3frq6	4frq6	5frq6
dmax1	1.000000	-0.795323	0.623580	0.425448	0.031063	0.535092	...	-0.100119	0.011025	-0.068843	-0.238951	-0.147716
dmin1	-0.795323	1.000000	-0.619117	-0.415592	-0.002960	-0.523460	...	0.122819	0.007684	0.098698	0.260335	0.158819
dstd1	0.623580	-0.619117	1.000000	0.946314	0.049078	0.970865	...	0.217360	0.402859	0.327938	0.170897	0.167545
damd1	0.425448	-0.415592	0.946314	1.000000	0.065187	0.946763	...	0.333912	0.533177	0.481438	0.341127	0.329089
dsma1	0.031063	-0.002960	0.049078	0.065187	1.000000	0.055171	...	-0.015141	0.080455	0.013974	0.007949	-0.021317
...	...	...	...	...	...	...	...	...	...		...	...
1frq6	-0.100119	0.122819	0.217360	0.333912	-0.015141	0.265381	...	1.000000	0.371936	0.388016	0.279830	0.305427
2frq6	0.011025	0.007684	0.402859	0.533177	0.080455	0.469896	...	0.371936	1.000000	0.605423	0.275870	0.166825
3frq6	-0.068843	0.098698	0.327938	0.481438	0.013974	0.369621	...	0.388016	0.605423	1.000000	0.450478	0.282635
4frq6	-0.238951	0.260335	0.170897	0.341127	0.007949	0.210590	...	0.279830	0.275870	0.450478	1.000000	0.419460
5frq6	-0.147716	0.158819	0.167545	0.329089	-0.021317	0.182000	...	0.305427	0.166825	0.282635	0.419460	1.000000

90 rows × 90 columns

Tabla 5.3.8: Correlación entre las variables.

Inicialmente se establece un valor de umbral de 0.85, pero se realiza un pequeño reajuste a 0.8642. Si la correlación entre un par de variables es mayor a este umbral significa entonces que presentan alta correlacion (>0.8642) y es posible descartar una de esas variables. Se extraen las columnas del conjunto de datos y el resultado se muestra en la tabla 5.3.9.

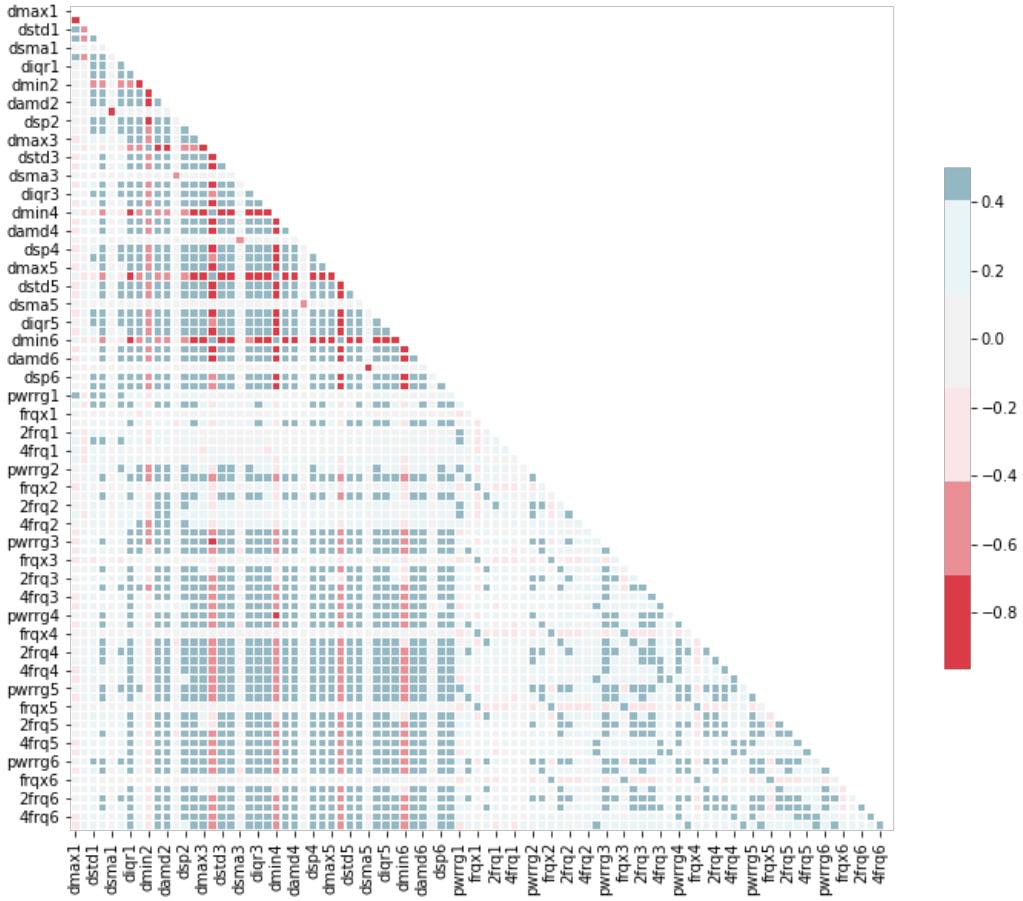


Figura 5.3.2: Matriz de correlación de los datos como mapa de calor.

```
# Seleccionar el triangulo superior de la matriz de correlacion
upper = dfx_corr.where(np.triu(np.ones(dfx_corr.shape), k=1).astype(np.bool))

# Buscar el indice de la columna de con valor de correlacion mayor al umbral

# reajustar umbral de 0.85 -> 0.8642 para redondear a 60 caracteristicas
to_drop = [column for column in upper.columns if any(upper[column] > 0.8642)]


# suprimir caracteristicas
df_drop=dfx_reduced.drop(dfx_reduced[to_drop], axis=1)
df_drop
```

	dmax1	dmin1	dstd1	dsma1	dmax2	dmin2	...	1frq6	2frq6	3frq6	4frq6	5frq6
dmax1	1.000000	-0.795323	0.623580	0.031063	-0.021685	0.017764	...	-0.100119	0.011025	-0.068843	-0.238951	-0.147716
dmin1	-0.795323	1.000000	-0.619117	-0.002960	0.021684	-0.006309	...	0.122819	0.007684	0.098698	0.260335	0.158819
dstd1	0.623580	-0.619117	1.000000	0.049078	0.433096	-0.446655	...	0.217360	0.402859	0.327938	0.170897	0.167545
dsma1	0.031063	-0.002960	0.049078	1.000000	0.002340	0.023594	...	-0.015141	0.080455	0.013974	0.007949	-0.021317
dmax2	-0.021685	0.021684	0.433096	0.002340	1.000000	-0.912880	...	0.273931	0.369854	0.344477	0.305696	0.319276
dmin2	0.017764	-0.006309	-0.446655	0.023594	-0.912880	1.000000	...	-0.249980	-0.345554	-0.338773	-0.340146	-0.333297
...	...	...	...	...	...	...	...	...	...	...	...	...
1frq6	-0.100119	0.122819	0.217360	-0.015141	0.273931	-0.249980	...	1.000000	0.371936	0.388016	0.279830	0.305427
2frq6	0.011025	0.007684	0.402859	0.080455	0.369854	-0.345554	...	0.371936	1.000000	0.605423	0.275870	0.166825
3frq6	-0.068843	0.098698	0.327938	0.013974	0.344477	-0.338773	...	0.388016	0.605423	1.000000	0.450478	0.282635
4frq6	-0.238951	0.260335	0.170897	0.007949	0.305696	-0.340146	...	0.279830	0.275870	0.450478	1.000000	0.419460
5frq6	-0.147716	0.158819	0.167545	-0.021317	0.319276	-0.333297	...	0.305427	0.166825	0.282635	0.419460	1.000000

560 rows × 60 columns

Tabla 5.3.9: Matriz de correlación después de aplicar un filtro de alta correlación.

Aplicando esta técnica se logra la reducción de 90 a 60 características.

El resultado de extraer la alta correlación se puede observar de mejor manera mediante el valor absoluto de la matriz de correlación como mapa de calor (figura 5.3.3).

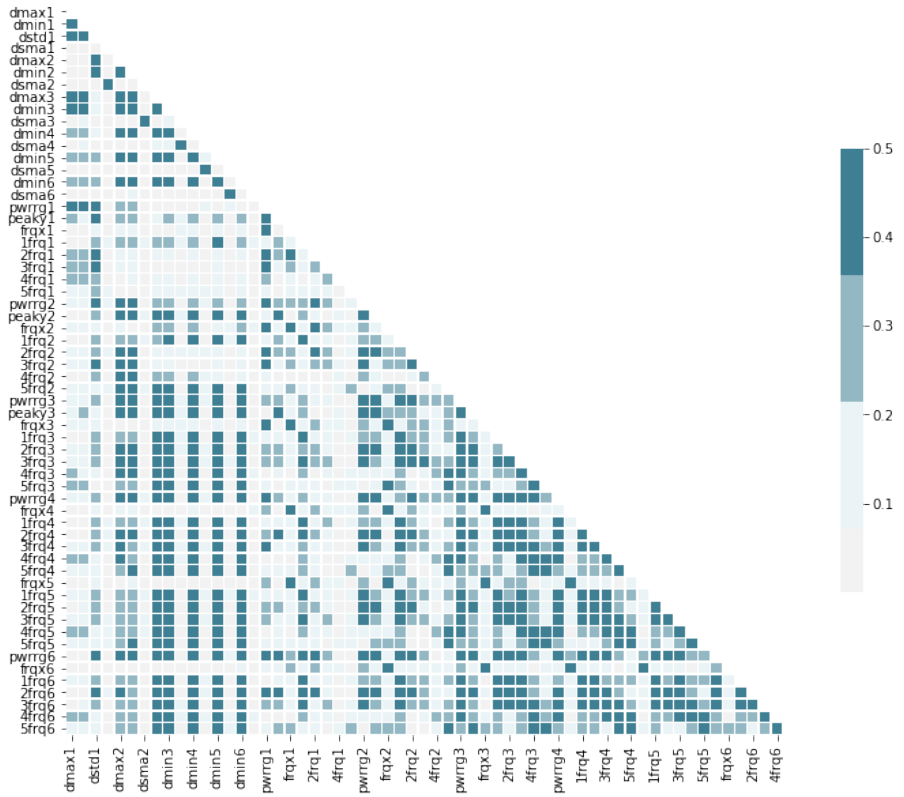


Figura 5.3.3: Matriz de correlación después de aplicar filtro de alta correlación.

5.3.3.1.4. Importancia de características

Indica la importancia relativa de cada característica al realizar una predicción. Con esto se puede tener una mejor comprensión del modelo.

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(random_state=1, max_depth=10)
df=pd.get_dummies(df_drop)
model.fit(df_drop,data.clas
features = df_drop.columns
importances = model.feature_importances_
indices = np.argsort(importances)[-60:]

plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='navy')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.grid()
plt.show()
```

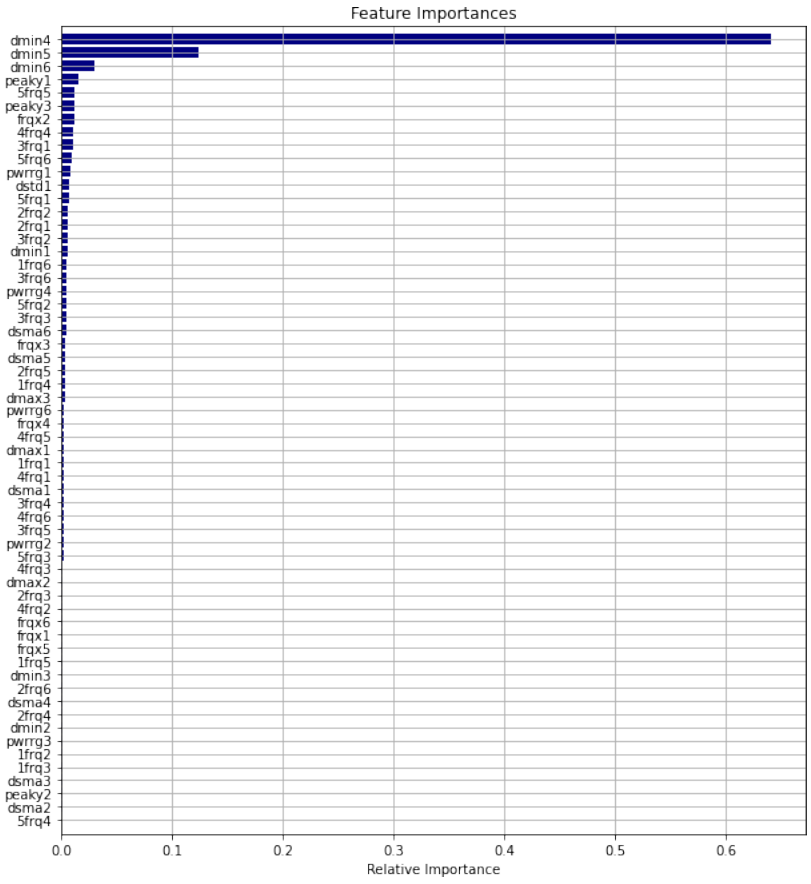


Figura 5.3.4: Relación de importancia de características.

De acuerdo a la gráfica de la figura 5.3.4 las 3 características principales son:

- *dmin4*, valor mínimo en el segmento 4.
- *dmin5*, valor mínimo en el segmento 5.
- *dmin6*, valor mínimo en el segmento 6.

Esto puede tener cierta relación con algún modo de propagación de las ondas superficiales o con la coda de la señal sísmica (Ortega, R. 2020). La señal segmentada se muestra en la figura 5.3.5 resaltando los segmentos 4, 5 y 6.

Se decide convenientemente dejar las 60 caracteristicas de entrada.

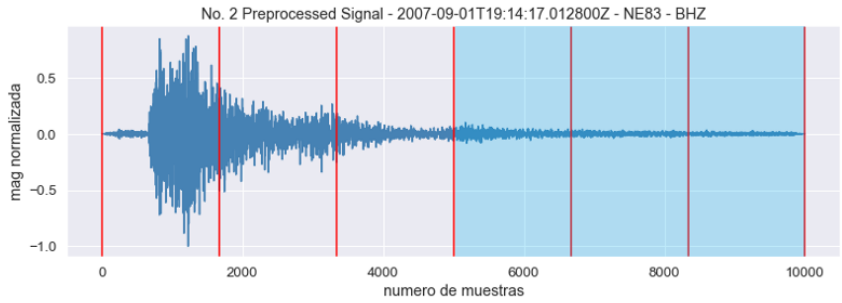


Figura 5.3.5: Señal sísmica segmentada mostrando los segmentos 4, 5 y 6.

5.3.3.2. Extracción de características

La extracción de características busca encontrar un conjunto más pequeño de nuevas variables, cada una de las cuales es una combinación entre esas mismas variables de entrada. En este caso, se aplicará la técnica de análisis de componentes principales.

5.3.3.2.1. Análisis de Componentes Principales.

El Analisis PCA (*Principal Component Analysis*) es un algoritmo de reducción de dimensionalidad muy útil como herramienta para la visualización, filtrado de ruido y extracción de características que permite simplificar la complejidad de espacios muestrales con muchas dimensiones conservando su información.

El objetivo principal de un análisis de PCA es detectar la correlación entre variables, si existe una fuerte correlación intenta reducir la dimensionalidad encontrando en los datos de alta dimensión

las direcciones de máxima variación y proyectarlo en un subespacio de dimensiones más pequeñas mientras se conserva la mayor parte de la información.

Pasos para implementar PCA.

Haciendo k= 60 dimensiones.

```
# 1. Calcular la matriz de covarianza a partir de datos estandarizados (de
    dimension d)
cov_mat_std = np.cov(df_drop.T)
print('NumPy covariance matrix: \n%s' %cov_mat_std)

# 2. Obtener los vectores y valores propios de la matriz de covarianza.
eig_vals_std, eig_vecs_std = np.linalg.eig(cov_mat_std)
print('Eigenvectors \n%s' %eig_vecs_std)
print('\nEigenvalues \n%s' %eig_vals_std)

# 3. Ordenar valores propios en orden descendente y eligir los k-vectores propios
     mas grandes
#  (autovector, autovalor)
eig_pairs = [(np.abs(eig_vals_std[i]), eig_vecs_std[:,i]) for i in range(len(
    eig_vals_std))]
# mostrar en orden descendente
eig_pairs.sort(key=lambda x: x[0], reverse=True)
print('Autovalores en orden descendente:')
for i in eig_pairs:
    print(i[0],'-',i[1])

# 4. Construir matriz de proyeccion W a partir de los k-vectores propios
    seleccionados. Para este ejemplo k=60.
matrix_w = np.hstack((eig_pairs[0][1].reshape(60,1),
eig_pairs[1][1].reshape(60,1)))
Y = df_drop.dot(matrix_w)
print('Matriz W:\n', matrix_w)

# Se construye una grafica con los k= 60 componentes.

tot_std = sum(eig_vals_std)
var_exp_std = [(i / tot_std)*100 for i in sorted(eig_vals_std, reverse=True)]
cum_var_exp_std = np.cumsum(var_exp_std)

with plt.style.context('seaborn-whitegrid'):
        plt.figure(figsize=(8, 6))
        plt.axvline(60, 0, 1, label='Components ~100% variance', color='red',
            linestyle='--')
        plt.bar(range(60), var_exp_std, alpha=0.5, align='center',
        label='Individual variance explained', color='navy')
        plt.step(range(60), cum_var_exp_std, where='mid', color='lightblue',
            label='Cumulative explained variance')
        plt.ylabel('Variance Ratio Explained')
        plt.xlabel('Main components')
        plt.title('Normalized')
        plt.legend(loc='best')
        plt.tight_layout()
```

En la gráfica de la figura 5.3.6 se muestra la varianza individual explicada y la varianza acumulada (de 99.11 %) con los 60 componentes.

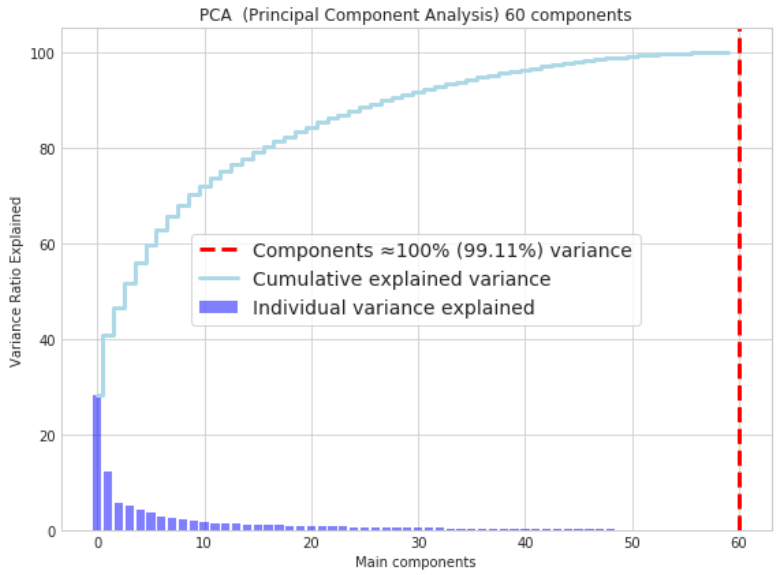


Figura 5.3.6: Análisis de componentes principales con *k*= 60 componentes.

Aplicar PCA con k=50 componentes.

Al aplicar PCA es recomendable conservar arriba de 95 %. Para lograr el 95 %, de acuerdo al listado de la Tabla 5.3.3 se requieren 41 características de entrada, sin embargo, en el desarrollo posterior el modelo pierde precisión, por lo que se aplicará análisis PCA con 50 componentes que explica al 97.38 % de la varianza.

Python posee una función predefinida para obtener PCA contenida en la libreria *scikit-Learn*. Al final se obtiene el conjunto de datos con 50 componentes de la tabla 5.3.10.

```
from sklearn.decomposition import PCA

k= 50  #  numero de componentes
pca_50=PCA(n_components=k)
x_pca_50 = pca_50.fit_transform(x_std)
dfx_pca_50 = pd.DataFrame(data = x_pca_50)
explained_pca_50 = pca_50.explained_variance_ratio_
dfx_pca_50
```

	0	1	2	3	4	5	...	44	45	46	47	48	49
0	2.775998	1.287781	10.395372	2.555141	1.220490	0.265059	...	0.847195	-1.158628	0.851426	0.912757	0.247787	-0.894961
1	-4.476349	4.154435	-0.642430	-1.293249	0.302045	1.323212	...	0.650439	-0.020414	0.151118	0.264395	0.237756	0.149949
2	-5.851072	4.597470	1.398824	2.823433	-0.625944	-3.650529	...	0.323641	-0.231529	-0.522079	-0.526275	0.102678	-0.610099
3	-0.664620	-1.696514	0.657350	0.622170	-0.841652	-1.954978	...	-0.173723	0.483719	0.383812	-0.378172	-0.334500	0.168586
4	-5.431556	3.149101	-0.061205	1.697248	-0.554422	-1.659356	...	-0.171978	-0.036530	-0.162832	0.100622	-0.068510	-0.112623
...	...	...	...	...	...	...	...	...	...	...	...	...	...
555	8.085056	4.057610	1.806434	0.646706	1.882012	1.100656	...	-0.495713	-0.230518	0.080259	0.732049	0.890032	0.593852
556	13.614976	2.614652	2.690288	-3.690919	-8.946732	4.205369	...	-0.446020	0.014514	-0.615988	-0.309500	0.290050	0.612798
557	1.589960	-2.448449	3.263704	-0.072590	-0.070895	-1.338734	...	-0.870982	0.442151	0.098397	0.351355	-0.150805	0.071855
558	7.654609	-3.718612	0.003442	-0.053163	-3.553531	-2.052065	...	0.812277	-0.456050	0.106379	-0.306709	0.431374	-0.633513
559	4.618455	3.212942	-0.492509	1.872358	-2.325113	-0.891536	...	0.266525	-2.066714	-0.376630	0.308184	-0.354509	-0.163838

Tabla 5.3.10: *Dataset* PCA con 50 componentes.

En la figura 5.3.7 se grafíca el acumulado de varianza explicada vs varianza acumulada en las nuevas dimensiones.

```
plt.figure(figsize=(8,6))
plt.plot(range(50), np.cumsum(pca_50.explained_variance_ratio_),'blue',linestyle='--',linewidth=3)
plt.plot(range(50), pca_50.explained_variance_ratio_,'red',linestyle='--', linewidth=3)

plt.title("Component-wise and Cumulative Explained Variance")
plt.xlabel('number of components')
plt.ylabel('explained variance')
plt.grid()
plt.yticks(np.arange(0, 1.01, 0.1))
plt.legend(('Cumulative Explained Variance','Component-wise'),
loc=(0.3, 0.44), handlelength=3.5, fontsize=14,frameon=True,framealpha=0.8)
plt.annotate('~97.38% explained variance', xy=(49,0.98), xytext=(0,0.93),fontsize=12,
arrowprops=dict(arrowstyle="->",
connectionstyle="angle3,angleA=0,angleB=5"));
plt.show()
```

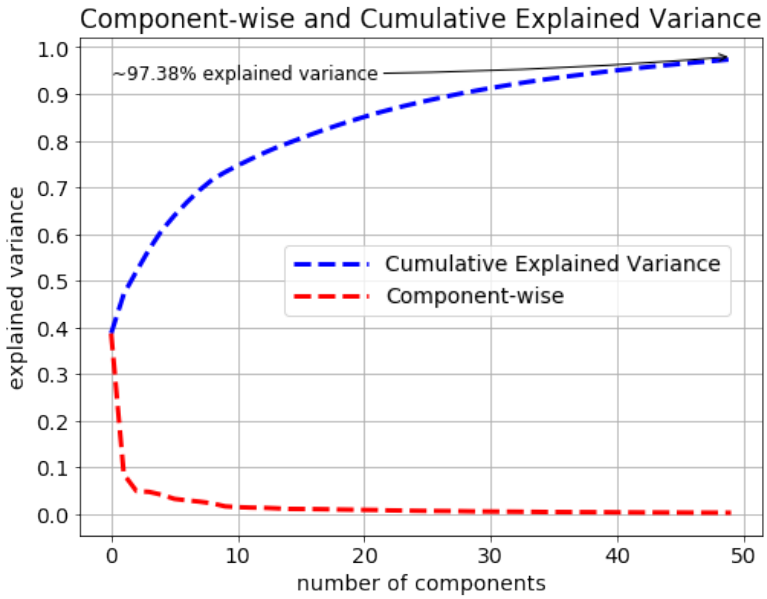


Figura 5.3.7: Componentes principales vs varianza explicada acumulada con PCA 50 componentes.

Varianza aportada de acuerdo al número de componentes

```
print('explained_variance_ratio (suma de componentes)')
print('variables\tvarianza expl\t acum\t\t %acum')
for i in range(k):
        print('%i\t\t %.6f\t%f\t (%.6f)' %(i+1,explained_pca_50[i], sum(
            explained_pca_50[0:i+1]), sum(explained_pca_50[0:i+1])*100))
        print('* con %d componentes se explica el %.2f%%.de la varianza' %(k,sum(
            explained_pca_50[0:k])*100))
```

razon de varianza explicada			
componentes	varianza explicada	varianza acumulada	% acumulado
1	0.386536	0.386536	(38.653601)
2	0.084557	0.471093	(47.109299)
3	0.048609	0.519702	(51.970249)
4	0.046963	0.566665	(56.666517)
5	0.040638	0.607303	(60.730295)
6	0.031885	0.639188	(63.918784)
7	0.028755	0.667943	(66.794276)
8	0.025957	0.693900	(69.389953)
9	0.022304	0.716203	(71.620315)
10	0.015930	0.732134	(73.213351)
11	0.014197	0.746331	(74.633057)
12	0.013266	0.759596	(75.959620)
13	0.012820	0.772416	(77.241616)
14	0.011721	0.784137	(78.413703)
15	0.010400	0.794537	(79.453715)
16	0.010255	0.804792	(80.479190)
17	0.010003	0.814795	(81.479538)
18	0.009658	0.824453	(82.445313)
19	0.009071	0.833524	(83.352395)
20	0.008702	0.842226	(84.222565)
21	0.008349	0.850575	(85.057468)
22	0.008081	0.858656	(85.865600)
23	0.007252	0.865908	(86.590843)
24	0.006944	0.872852	(87.285247)
25	0.006446	0.879298	(87.929831)
26	0.006178	0.885476	(88.547594)
27	0.005926	0.891402	(89.140157)
28	0.005775	0.897177	(89.717684)
29	0.005375	0.902552	(90.255163)
30	0.004951	0.907503	(90.750252)
31	0.004810	0.912312	(91.231207)
32	0.004654	0.916966	(91.696636)
33	0.004503	0.921469	(92.146942)
34	0.004240	0.925710	(92.570986)
35	0.003912	0.929622	(92.962223)
36	0.003753	0.933376	(93.337559)
37	0.003651	0.937026	(93.702612)
38	0.003506	0.940532	(94.053186)
39	0.003419	0.943951	(94.395058)
40	0.003347	0.947298	(94.729778)
41	0.003103	0.950401	(95.040052)
42	0.002977	0.953378	(95.337767)
43	0.002828	0.956206	(95.620579)
44	0.002779	0.958985	(95.898521)
45	0.002679	0.961664	(96.166396)
46	0.002624	0.964288	(96.428796)
47	0.002465	0.966753	(96.675271)
48	0.002376	0.969129	(96.912858)
49	0.002333	0.971462	(97.146167)
50	0.002308	0.973769	(97.376932)
* con 50 componentes se explica el 97.38 % de la varianza			

Tabla 5.3.11: Varianza explicada y acumulada con PCA 50 componentes.

Dataset

Después de aplicar las técnicas de reducción de dimensionalidad, se genera un archivo csv con los 50 componentes principales el cual se muestra en la tabla 5.3.12.

```
dfy = pd.DataFrame(data=y)
dataset_pca= pd.concat([dfy,dfx_pca], axis=1)
dataset_pca.to_csv('data_pca50.csv')
dataset_pca
```

	clase	pc01	pc02	pc03	...	pc47	pc48	pc49	pc50
0	1	2.775998	1.287781	10.395372	...	0.851426	0.912757	0.247787	-0.894961
1	1	-4.476349	4.154435	-0.642430	...	0.151118	0.264395	0.237756	0.149949
2	1	-5.851072	4.597470	1.398824	...	-0.522079	-0.526275	0.102678	-0.610099
3	1	-0.664620	-1.696514	0.657350	...	0.383812	-0.378172	-0.334500	0.168586
4	1	-5.431556	3.149101	-0.061205	...	-0.162832	0.100622	-0.068510	-0.112623
...	...	...	...	...	...	...	...	...	...
555	0	8.085056	4.057610	1.806434	...	0.080259	0.732049	0.890032	0.593852
556	0	13.614976	2.614652	2.690288	...	-0.615988	-0.309500	0.290050	0.612798
557	0	1.589960	-2.448449	3.263704	...	0.098397	0.351355	-0.150805	0.071855
558	0	7.654609	-3.718612	0.003442	...	0.106379	-0.306709	0.431374	-0.633513
559	0	4.618455	3.212942	-0.492509	...	-0.376630	0.308184	-0.354509	-0.163838

560 rows × 51 columns

Tabla 5.3.12: Dataset después de aplicar técnicas de reducción de dimensionalidad.

Resumen del preprocesamiento

En resumen, al conjunto de datos con 120 características se le aplicaron técnicas de reducción de dimensionalidad, mediante selección de características se reduce a 60 características, y con extracción de características a 50 componentes principales. Gráficamente se muestra en la figura 5.3.8.

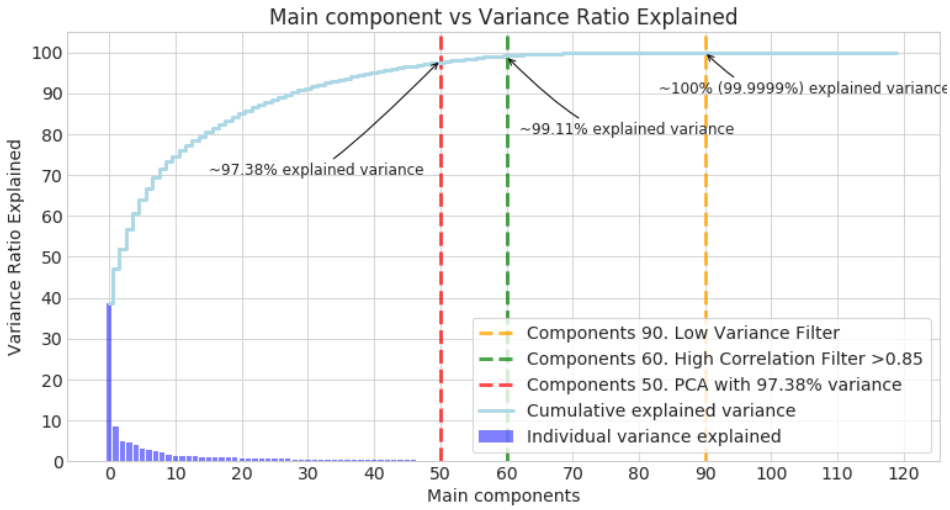


Figura 5.3.8: Resumen de aplicar reducción de dimensionalidad al conjunto de datos.

5.3.4. Red Neuronal Artificial con reducción de dimensionalidad

5.3.4.1. Leer el *dataset*, normalizar y dividir en datos de entrenamiento y de prueba

El nuevo dataset de entrada contiene 50 componentes con 560 muestras las cuales se normalizan y se dividen en 70 % (392) para muestras de entrenamiento y el 30 % (168) para prueba.

5.3.5. Definir el diccionario de hiperparámetros

Se define el diccionario de hiperparámetros:

```
p = {
        'first\_neuron':        (10,150,14),
        'hidden_layers':        [1,2],
        'hidden_neuron':        (10,200,19),
        'batch_size':           [16, 32, 64, 128, 256],
        'loss':                 ['binary_crossentropy'],
        'optimizer':            ['adam', 'RMSprop', 'adamax'],
        'kernel_initializer':   ['uniform','normal'],
        'epochs':               [300],
        'last_activation':      ['softmax', 'sigmoid']
}
```

5.3.6. Construir el modelo

```
def model_NN(x_train, y_train, x_test, y_test, params):
        model = Sequential()

        # primera capa
        model.add(Dense(params['first_neuron'], input_dim= x_train.shape[1],
           activation='relu', kernel_initializer = params['kernel_initializer'])
           )

        # genera capas ocultas
        for i in range(params['hidden_layers']):
                print (f"adding layer {i+1}")
                model.add(Dense(params['hidden_neuron'],activation='relu',
                   kernel_initializer=params['kernel_initializer']))

        # capa de salida
        model.add(Dense(2, activation=params['last_activation'],
        kernel_initializer=params['kernel_initializer']))

        # se compila el modelo
        model.compile(loss= params['loss'],optimizer=params['optimizer'], metrics
           =['acc'])
```

```
18
19          # entrenamiento
20          history = model.fit(x_train, y_train,validation_data=[x_test, y_test],
                batch_size=params['batch_size'],epochs=params['epochs'], callbacks=[
                early_stopper(params['epochs'], patience=20)],verbose=0)
21
22          return history, model
```

5.3.6.1. Configurar parámetros del experimento

En la tabla 5.3.13 se definen los parametros para el experimento.

Parámetro	**Valor**	**Descripción**
x	x_train	Características de predicción
y	y_train	Variable de resultados de la predicción
x_val	x_test	Datos de validación para x
y_val	y_test	Datos de validación para y
model	model_NN	Modelo de Keras
params	p	Diccionario de parámetros
print_params	True	Imprime los hiperparámetros de cada permutación
round_limit	1000	Número de permutaciones a ejecutar
experiment_name	NNscan1000	Genera una carpeta con este nombre y salva los resultados del experimento
clear_session	True	Borra sesión de backend entre permutaciones
save_weights	True	Salva los pesos del modelo
reduction_metric	'val_acc'	Métrica que se utilizará para la reducción.
minimize_loss	True	Minimizar perdida

Tabla 5.3.13: Parámetros del experimento con Talos con reducción de dimensionalidad.

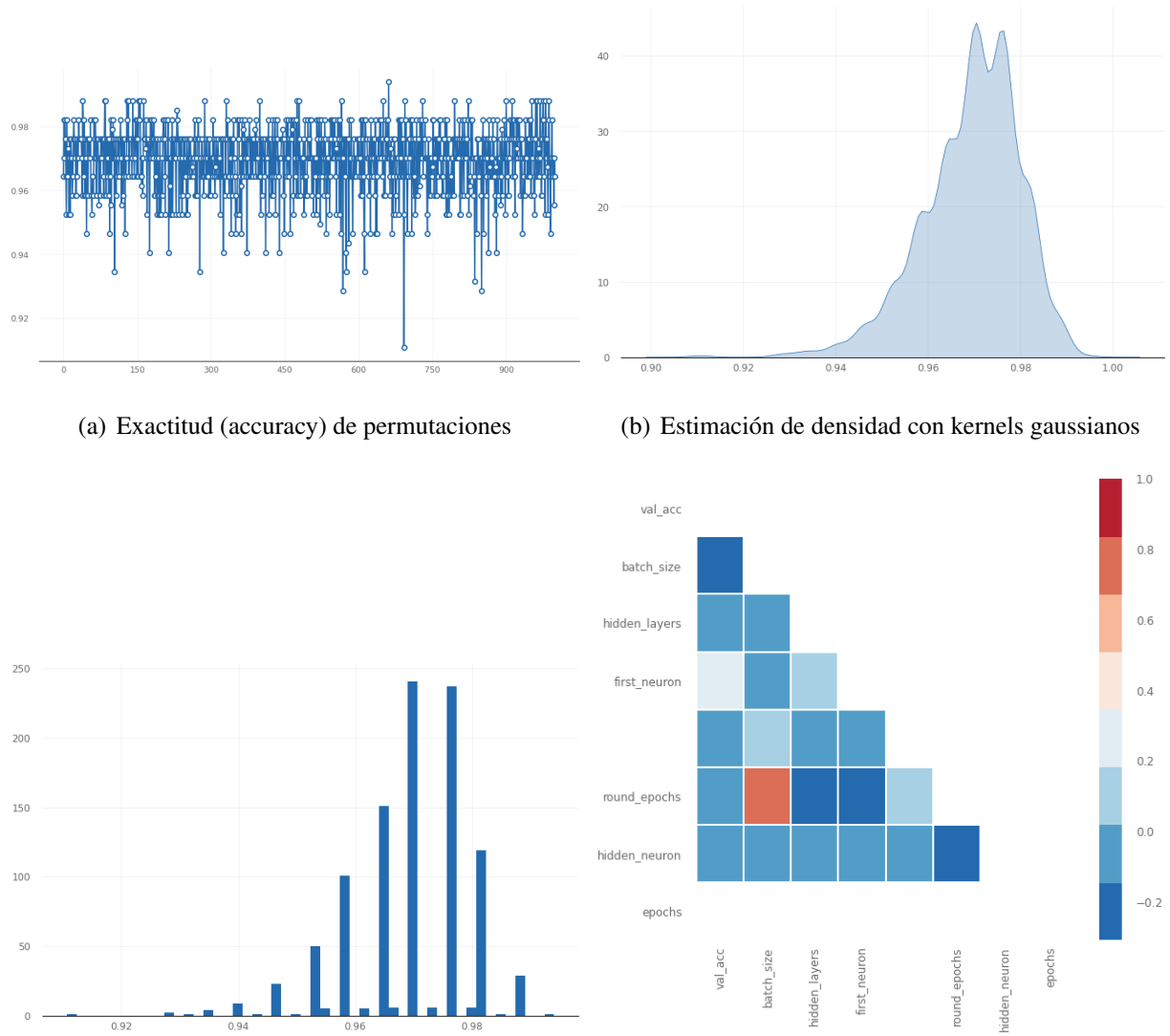
5.3.6.2. Ejecutar experimento

Se ejecuta el experimento con la configuración definida..

```
1  start\_time = time.time()
2  t = ta.Scan( x=x\_train,
3          y=y\_train,
4          x\_val= x\_test,
5          y\_val= y\_test,
6          model=model\_NN,
7          params=p,
8          print\_params= True,
9          round\_limit= 1000,
10         experiment\_name= experiment,
```

```
        clear\_session= True,
        save\_weights= True,
        reduction\_metric= 'val\_acc',
        minimize\_loss= True )
print(t.data)
end\_time = time.time()
print("\n--- (%s segundos) ---" % (end_time - start_time)
```

Una vez generada las permutaciones,se puede acceder a las configuraciones dentro del espacio de hiperparámetros y mostrar algunas de las gráficas como se muestran en la figura 5.3.9.



(a) Exactitud (accuracy) de permutaciones

(b) Estimación de densidad con kernels gaussianos

(c) Histograma de las permutaciones generadas

(d) Correlación en mapa de calor

Figura 5.3.9: Gráficas generadas dentro del espacio de hiperparámetros para el conjunto de datos con reducción de dimensionalidad.

5.3.7. Evaluación de los modelos

Se evalúa el rendimiento del modelo contra una validación cruzada con k=10.

```
evaluate_object = ta.Evaluate(t)
evaluate_object.evaluate(x_test, y_test, folds=10, metric='val_acc', task='
    multi_label')
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9372549019607843, 1.0, 1.0, 1.0]
```

5.3.8. Implementación del modelo

Se implementa el método *Deploy()*, con el modelo generado en función de la métrica '*val_acc*'.

5.3.9. Restauración del modelo

Se restauran los archivos del experimento contenidos en el archivo de implementacion *deploy*:

- **_deploy_model.h5** que contiene los pesos de la red neuronal. HDF5 es un formato de datos jerárquico, se utiliza como formato de almacenamiento de datos por ser flexible, muy conveniente para almacenar matrices de valores reales, como los generados para los pesos de redes neuronales.
- **_deploy_model.json** que contiene la estructura del modelo de acuerdo a la métrica elegida. JSON es un formato utilizado para describir datos jerárquicamente. Keras tiene la capacidad de describir cualquier modelo usando el formato JSON.

5.3.10. Cargar el modelo y analizar resultados

El modelo obtenido de acuerdo a la métrica *val_accu* se muestra en la tabla 5.3.14:

acc	1.0
loss	1.08307e-3
val_acc	0.994048
val_loss	0.0534851
first_neuron	40
hidden_layers	2
hidden_neuron	140
batch_size	128
loss_function	binary_crossentropy
optimizer	adamax
kernel_initializer	uniform
last_activation	softmax

Tabla 5.3.14: Arquitectura de la red neuronal.

5.3.11. Informe de las principales tasas y métricas de clasificación

El reporte de las principales tasas y métricas de clasificacion obtenidas con el modelo se muestran en las tablas 5.3.15 y 5.3.16.

Accuracy:	99.40 % (0.994048)
Precision:	98.89 % (0.988889)
Recall:	100.00 % (1.000000)

Tabla 5.3.15: Métricas de clasificación.

classification_report

	precision	recall	f1-score	support
0	0.99	1.00	0.99	79
1	1.00	0.99	0.99	89
accuracy			0.99	168
macro avg	0.99	0.99	0.99	168
weighted avg	0.99	0.99	0.99	168

Tabla 5.3.16: Reporte de clasificación.

Errores de clasificación

De acuerdo a la tabla 5.3.16, se comete un error de clasificación, 1 falso negativo en la muestra 84 (tabla 5.3.17)

#	index	y_real	y_pred	clasificación
1	84	0	1	falso negativo x
84	NE74	2007-09-19T13:37:01.009300Z		
Total= 0 falso(s) positivo(s) 1 falso(s) negativo(s)				

Tabla 5.3.17: Errores de clasificación.

La forma de onda de registro mal clasificado se muestra en la figura 5.3.10.

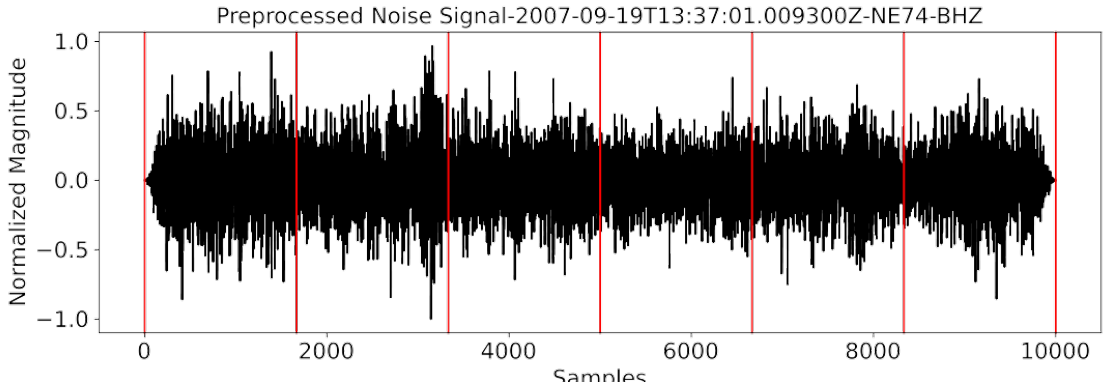


Figura 5.3.10: Forma de onda del registro mal clasificado con la red neuronal con PCA 50 componentes.

Matriz de confusión

La matriz de confusión se muestra en la tabla 5.3.18 y en la figura 5.3.11

Pérdida logarítmica

Best Log-Loss (validation): 0.03626319393515587

Minimizar log loss es equivalente a maximizar la precisión del clasificador. log loss debe estar cerca de cero para un buen algoritmo predictivo.

	Predicted Seismo	Predicted Noise
Real Seismo	89	0
Real Noise	1	78

True Positives TP:	89
False Positives FP:	0
True Negatives TN:	1
False Negatives FN:	78

Tabla 5.3.18: Matriz de confusión.

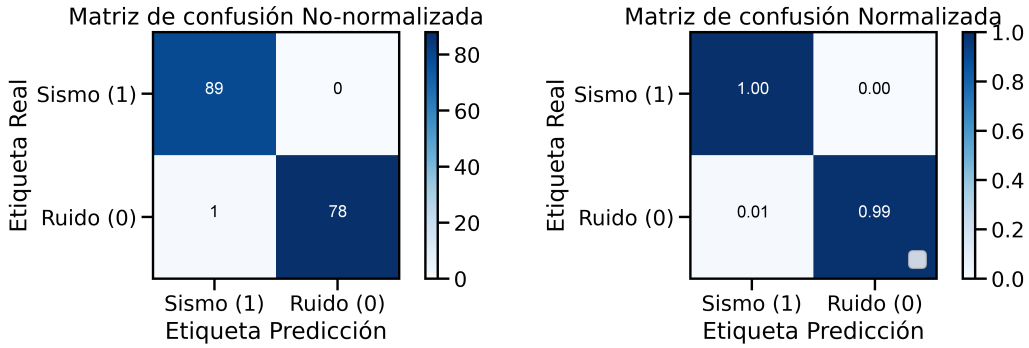


Figura 5.3.11: Matriz de confusión.

Coeficiente de determinación R²

Coeficiente de determinación (R^2): 0.976105817095719

Un valor cercano a 1 indica un modelo con un error cercano a cero, y un valor cercano a cero indica un modelo muy cercano a la línea de base.

Curva ROC / AUC

Porcentaje de AUC Área bajo la curva ROC.

AUC: 0.998293 AUC-*Area Under the ROC Curve. Área bajo la curva ROC*

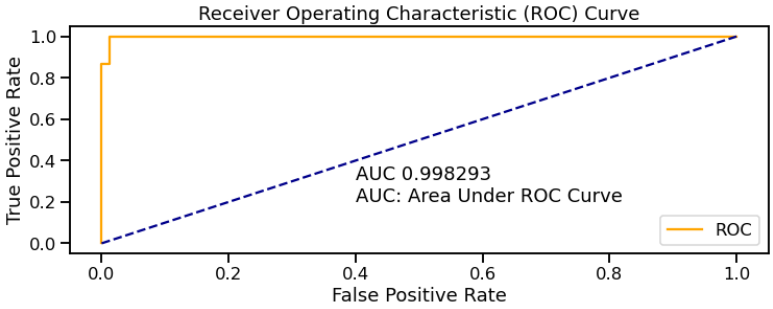


Figura 5.3.12: Área bajo la curva ROC.

5.3.12. Arquitectura y rendimiento del modelo obtenido con Reducción de dimensionalidad PCA=50

```
Model: "sequential_1
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 40)                2040
_________________________________________________________________
dense_2 (Dense)              (None, 140)               5740
_________________________________________________________________
dense_3 (Dense)              (None, 140)               19740
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 282
=================================================================
Total params: 27,802
Trainable params: 27,802
Non-trainable params: 0
```

Tabla 5.3.19: Capas y número de neuronas de la red neuronal con 50 componentes.

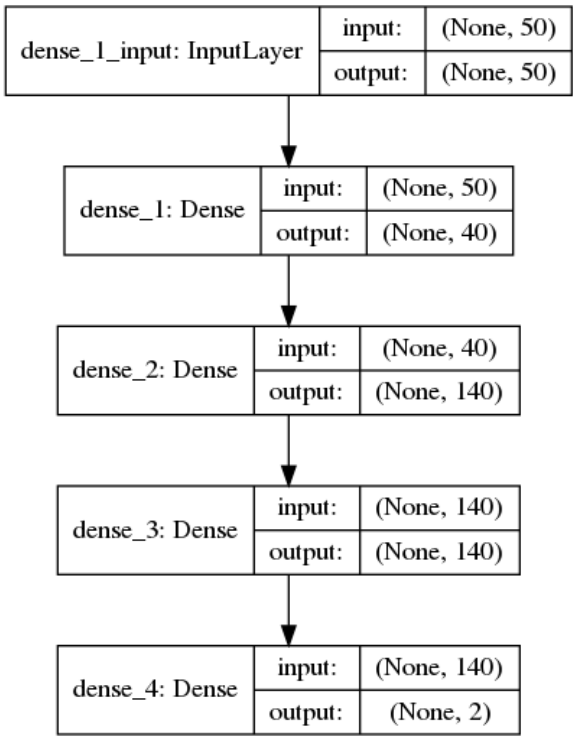
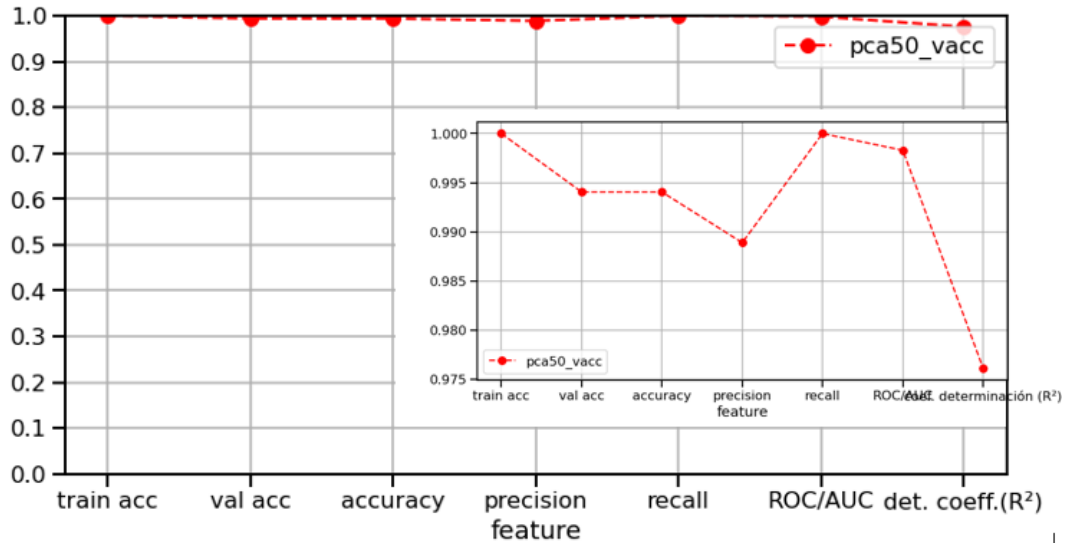


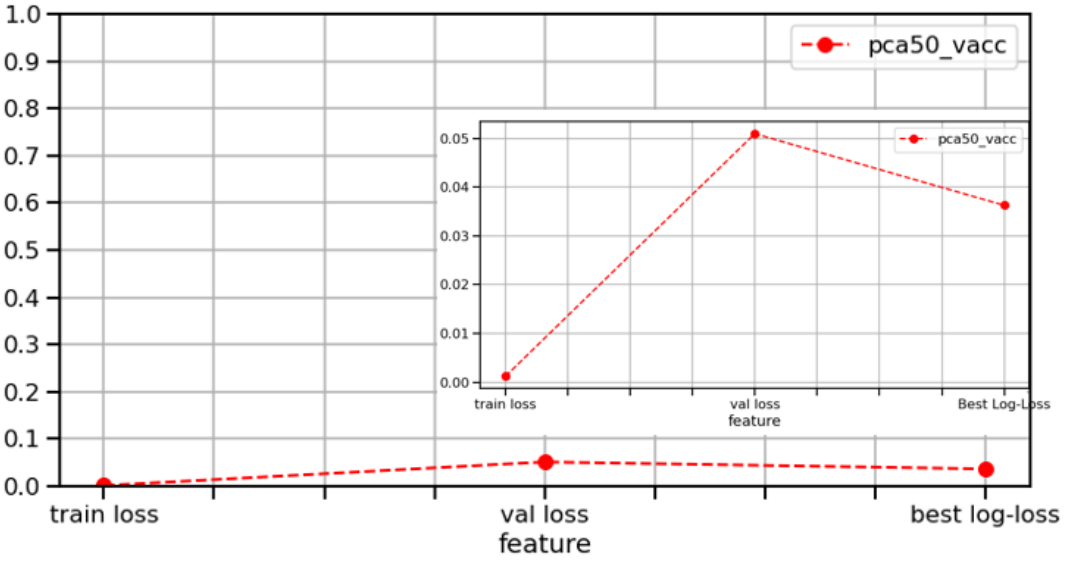
Figura 5.3.13: Distribución de capas de la red neuronal con reducción de dimensionalidad PCA=50.

Feature	**pca50_vacc**
archivo	scan_pca50_val_acc
first_neuron	40
hidden_layers	2
hidden_neuron	140
batch_size	128
loss_function	binary_crossentropy
optimizer	Adamax
kernel_initializer	uniform
last_activation	softmax
train acc	1
train loss	0.00120232
val acc	0.994048
val loss	0.0509395
accuracy	0.994048
precision	0.988889
recall	1
ROC/AUC Area Under the Curve	0.998293
best log-loss	0.0362632
coeficiente de determinación (R^2)	0.976106
False Positive FP	0
False Negative FN	1

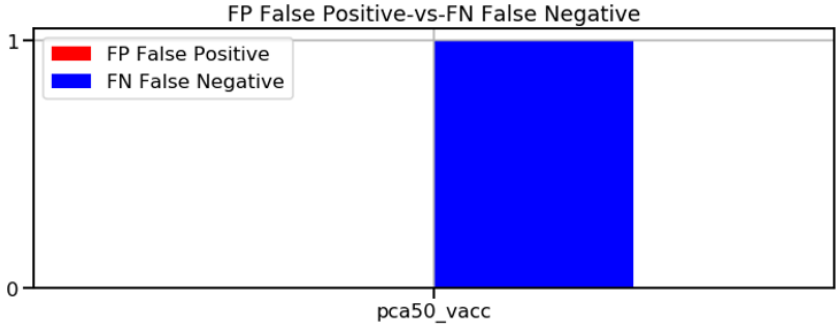
Tabla 5.3.20: Arquitectura y rendimiento del modelo de Red Neuronal Artificial con PCA 50 componentes.



(a) Rendimiento del modelo



(b) Pérdidas del modelo



(c) Errores de clasificación

Figura 5.3.14: Rendimiento del modelo de red neuronal artificial con PCA 50 componentes

Representación gráfica de la Red Neuronal Artificial con PCA 50 componentes

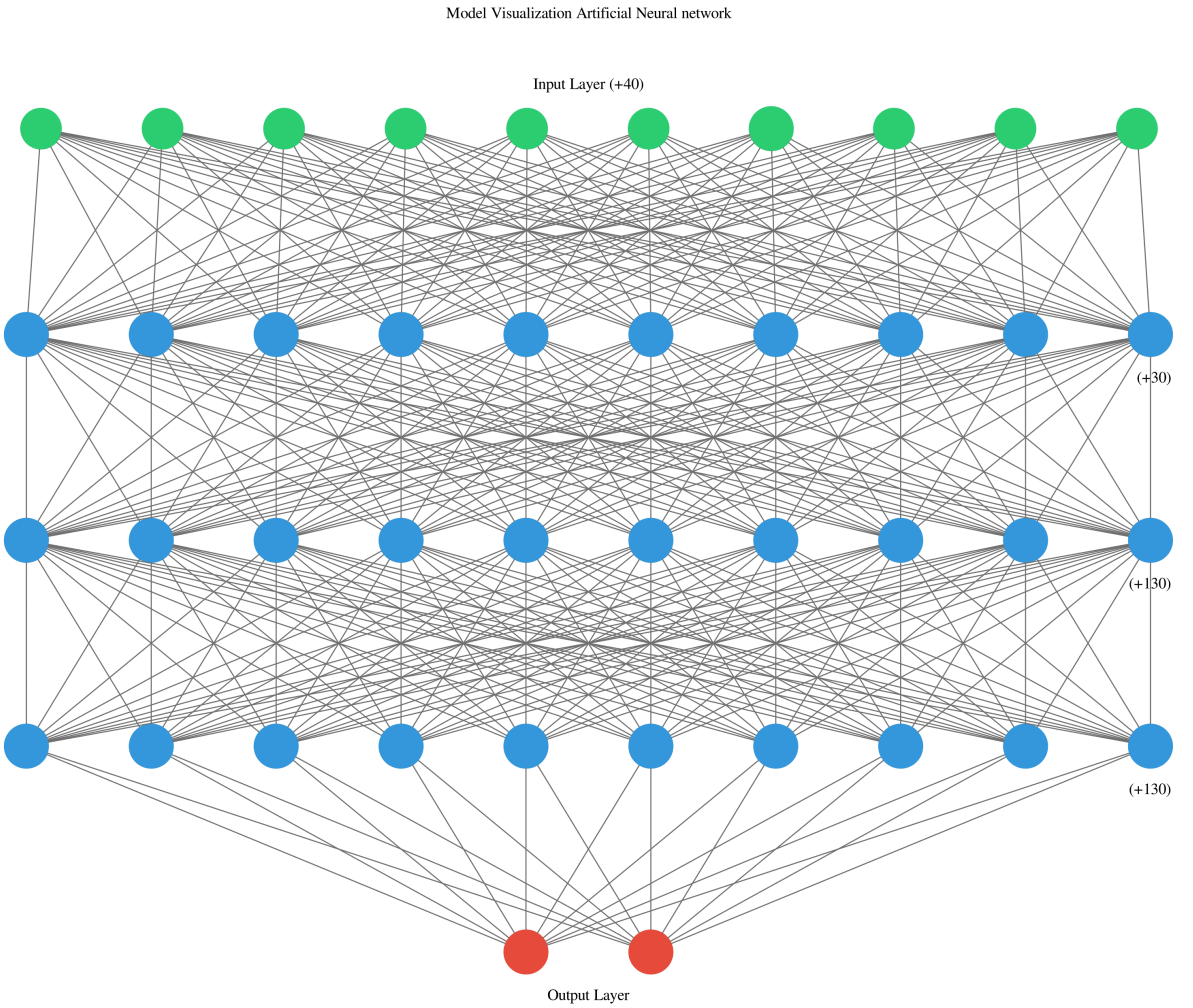
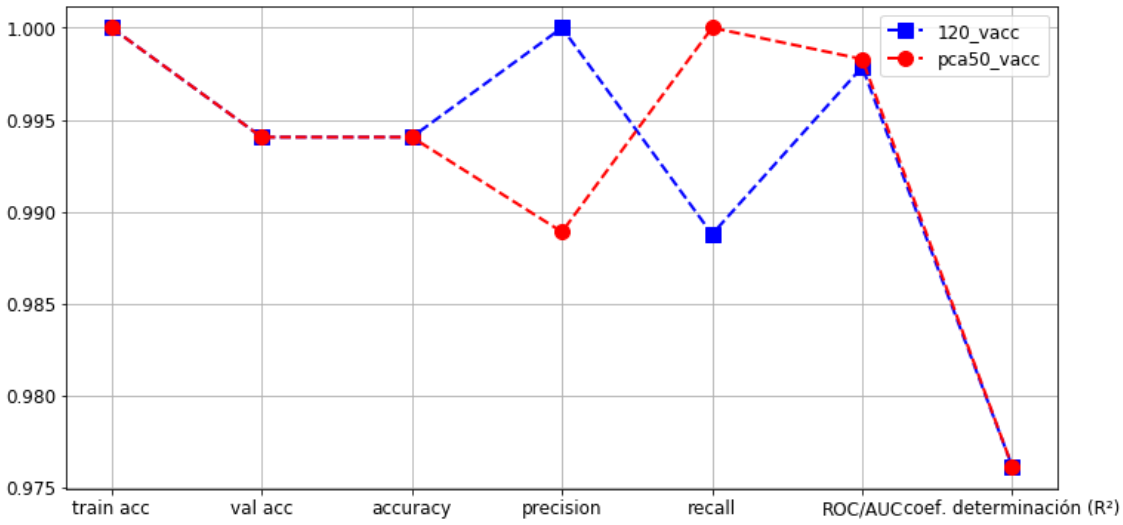


Figura 5.3.15: Representación gráfica de la red neuronal artificial con PCA 50 componentes.

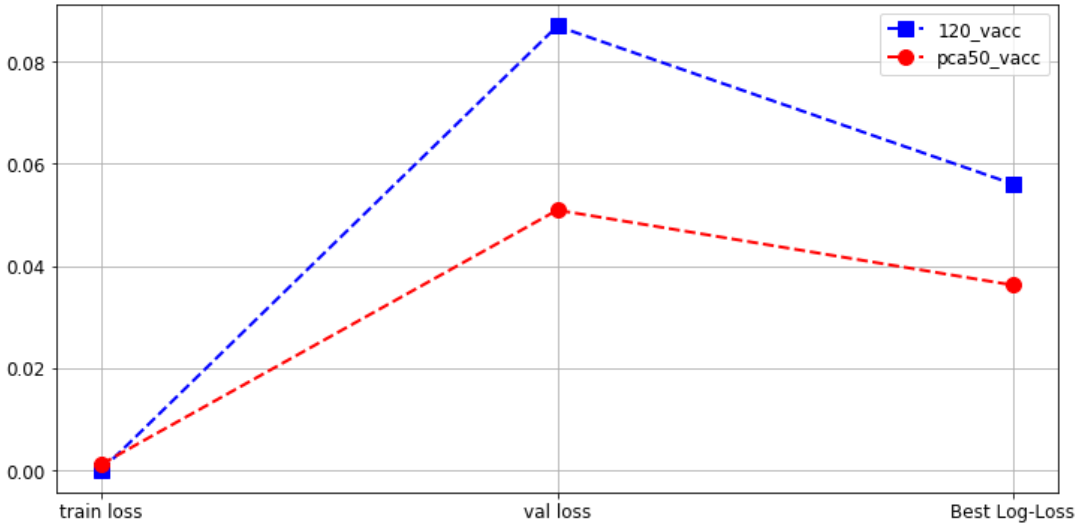
5.4. Resultados de la Red Neuronal Artificial con 120 características vs modelo con PCA 50 componentes

Features	**120_vacc**	**pca50_vacc**
archivo	NNscan500_val_acc	scan_pca50_val_acc
first_neuron	100	40
hidden_layers	2	2
hidden_neuron	270	140
batch_size	16	128
loss_function	binary_crossentropy	binary_crossentropy
optimizer	adam	Adamax
kernel_initializer	normal	uniform
last_activation	softmax	softmax
train acc	1.0	1.0
train loss	2.5232123770903704e-05	0.0012023200147918292
val acc	0.9940476417541504	0.9940476417541504
val loss	0.08698600216367984	0.050939528892437615
accuracy	0.9940476190476191	0.9940476190476191
precision	1.0	0.9888888888888889
recall	0.9887640449438202	1.0
ROC/AUC Area Under the Curve	0.9978665908121178	0.9982932726496941
best Log-Loss	0.05607054618719433	0.03626319393515587
coeficiente de determinación (R^2)	0.976105817095719	0.976105817095719
false positive FP	1	0
false negative FN	0	1

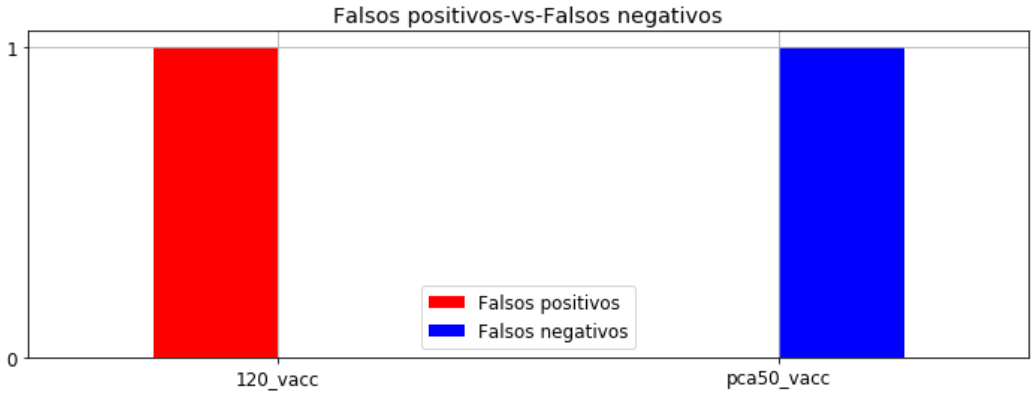
Tabla 5.4.1: Comparación de arquitectura y rendimiento de los modelos de Red Neuronal 120 características vs PCA=50.



(a) Gráfica del rendimiento



(b) Gráfica de las pérdidas



(c) Gráfica de errores de clasificación

Figura 5.4.1: Gráficas del rendimiento de los modelos de redes neuronales 120 características vs PCA 50 componentes.

5.5. Análisis de resultados

Para la identificación automática de regístros sísmicos se utilizaron técnicas de reconocimiento de patrones mediante la extracción de características y de una red neuronal artificial. Se dividieron los registros sísmicos en 6 segmentos no superpuestos del cual se extrajeron 20 características por segmento: 9 en el dominio del tiempo, 11 en el dominio de la frecuencia haciendo un total de 120 características por registro. Se establecen 2 experimentos definiendo la red neuronal con las 120 características de entrada y el otro experimento realizando una reducción de dimensionalidad de las características.

Ningun experimento será 100 % seguro, siempre existe la posibilidad de llegar a una conclusión en la que se pueden cometer estos dos tipos de errores: tipo I o Falso Positivo y tipo II o Falso Negativo. Lo que si resulta crucial es comprender la importancia de estos dos tipos de errores y lo que cada uno representa. Los riesgos de estos dos errores están inversamente relacionados y se determinan según el nivel de importancia, por lo que se debe determinar qué error tiene consecuencias más graves para una situación determinada antes de definir los riesgos.

Como se muestra en la tabla 5.5.1, la matriz de confusión para este modelo tiene etiquetadas las filas con las clases reales, y las columnas, con las clases pronosticadas por el modelo. Las clases reales en la matriz de confusión son positivas o etiquetadas como “clase 1” cuando el registro es un sismo y negativas o “clase 0” cuando se trata de ruido.

		Predicción del Modelo	
		“Sismo” (Clase 1)	“Ruido” (Clase 0)
Clase Real	“Sismo” (Clase 1)	Positivos Reales	Falsos Positivos
	“Ruido” (Clase 0)	Falsos Negativos	Negativos Reales

Figura 5.5.1: Matriz de confusión binaria.

En la matriz de confusión binaria la diagonal principal contendra la suma de todas las predicciones

correctas: los positivos reales y negativos reales. La otra diagonal refleja los errores del clasificador: los falsos positivos o errores tipo I y los falsos negativos o errores tipo II.

El algoritmo en la clasificación de los errores que se cometen en la matriz de confusión es el siguiente:

a. Falso Positivo o error tipo I. El registro contiene un sismo, pero el algoritmo pronostica que no contiene un sismo, que se trata de ruido. El impacto es que dejaran de etiquetarse sismos cuando el propósito es de encontrarlos en los registros.

b. Falso Negativo o error tipo II. El registro no contiene un sismo, se trata de ruido, pero el algoritmo predice que si existe un sismo.

El resultado final de los experimentos fue el siguiente:

- En el modelo obtenido con 120 características se logra un error de clasificación de 1 Falso Positivo (error de tipo I).

- Para el modelo obtenido aplicando reducción de dimensionalidad se presenta un error de clasificación de 1 Falso Negativo (error de tipo II).

Al tratarse de un modelo de clasificación binaria que busca clasificar la forma de onda en un registro sísmico, resulta mas costoso un error tipo I un Falso Positivo clasificar como ruido un sismo y perderlo, que señalar que una señal de ruido sismico es un sismo, ya que a pesar de tratarse de ruido, no se perdería su identificación y análisis, es decir, se dejaría de clasificar eventos sismicos cuando el propósito del sistema automatizado es clasificarlos.

Capítulo 6

CONCLUSIONES

La conclusión final en el desarrollo de la red neuronal artificial es que resulta “menos malo” clasificar errores tipo II o falsos negativos que errores tipo I o falsos positivos. Un falso Negativo o error tipo II se traduce que clasificar registros que contienen ruido como sismos, pero no se “escaparán” de ser etiquetados los sismos.

El modelo de red neuronal al que se le aplicó reducción de dimensionalidad: con 50 neuronas de entrada, 1 primera capa de 40 neuronas y 2 capas ocultas de 140 resulto el más eficiente, consume menos recursos computacionales y obtiene una eficiencia del 99 %, llega a superar por mucho a las realizadas por analistas humanos, cometiendo solo 1 error de clasificacion tipo II (Falso Negativo).

Los resultados de este estudio están incluidos en la sección “Development of an artificial neural network” del artículo “Insights of the Cerralvo Earthquake-Hurricane Henriette crisis in La Paz, México: Aftershocks detection with artificial neural networks.” Roberto Ortega PhD, Dana Carciumaru PhD, Alfredo Aguirre Msc, Israel Santillan PhD, Saul Martinez PhD [3].

El desarrollo de una red neuronal artificial para reconocer de manera más eficiente las réplicas de terremotos resultó ser una herramienta muy poderosa en situaciones de crisis como el evento Cerralvo-Henriette ocurido en Septiembre de 2007 .

6.1. Trabajo futuro

Dentro del trabajo que podría estar pendiente se puede mencionar:

- Realizar una interfase amigable con el usuario que puede no estar relacionado con el desarrollo del sistema.
- Implementar todas las etapas incluidas en el desarrollo del sistema incluyendo algunas que se realizaron de manera manual a fin de automatizar todo el proceso.
- Incluir las rutinas para las peticiones de los registros a los centros sismicos proporcionadas por el Dr. Roberto Ortega.

Bibliografía

[1] Jan T. Kozak and Charles D. James. Historical Depictions of the 1755 Lisbon Earthquake, 1998.

[2] Juan Murria. El terremoto de Lisboa del 1o de noviembre de1755: ¿El primer desastre "moderno"? *EIRD Estrategia Internacional para la Reducción de Desastres. Reducción de Desastres en las Américas*, 14, 2007.

[3] Roberto Ortega, Dana Carciumaru, Alfredo Aguirre, Israel Santillan, and Saúl Martínez. Insights of the September 2007 Cerralvo Earthquake–Hurricane Henriette Crisis in La Paz, Mexico: Aftershocks Detection with Artificial Neural Networks. *SRL Seismological Research Letters*, 2020.

[4] John G. Proakis and Dimitris Manolakis. *Tratamiento Digital de Señales*. Pearson. Prentice, 4th editio edition, 2011.

[5] Jens Havskov and Gerardo Alguacil. *Instrumentation in earthquake seismology*. Springer International Publishing AG, 2nd editio edition, 2015.

[6] E.J. Tarbuck, F.K. Lutgens, and D. Tasa. *Ciencias de la Tierra. Una introducción a la geología física.* Pearson Prentice Hall, 8a edicion edition, 2005.

[7] Dhananjay Kumar and Imtiaz Ahmed. Seismic Noise. In *Encyclopedia of Solid Earth Geophysics*, pages 1–6. 2020.

[8] NARS-Baja Seismic Network. NARS-Baja Seismic Network.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer International Publishing AG, 2011.

[10] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. O´Reilly, 2nd editio edition, 2019.

[11] Jake VanderPlas. *Python Data Science Handbook: Essential Tools for Working with Data*. 2016.

[12] Julian Avila and Trent Hauck. *Scikit-learn Cookbook: Over 80 recipes for machine learning in Python with scikit-learn*. Packt Publishing, Birminham- Mumbai, 2017.

[13] Gavin Hackeling. *Mastering Machine Learning with scikit-learn*. Publishing, Packt, 2014.

[14] Richard P. Lippmann. An introduction to computing with neural nets. *ACM SIGARCH Computer Architecture News*, 16(1):7–25, 1988.

[15] Wesam Salah Alaloul and Abdul Hannan Qureshi. Data processing using neural networks. *Department of Civil and Environmental Engineering, Universiti Teknologi Petronas, Perak, Malaysia*, 277(2):273–287, 1993.

[16] Jason Brownlee. *Develop Deep Learning Models On Theano And TensorFlow Using Keras*, volume 53. Machine Learning Mastery, 2019.

[17] Antonio Gulli and Sujit Pal. *Deep Learning with Keras*. Publishing, Packt, 2017.

[18] Jason Brownlee. How to Configure the Number of Layers and Nodes in a Neural Network.

[19] Keras Team. Keras: The Python Deep Learning library.

[20] Jojo Moolayil. *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python*. Apress, 2019.

[21] Navin Kumar Manaswi. *Deep Learning with Applications Using Python: Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras*. Apress, 2018.

[22] François Chollet. *Deep Learning with Python*. Manning Publications Co, 2018.

[23] Mikko Kotila. Talos Autonomy: Hyperparameter Optimization for Keras.

[24] David Cournapeau. Preprocessing data with scikit-learn 0.24.1.

[25] Data Flair. What is Dimensionality Reduction – Techniques, Methods, Components, 2020.

[26] Robert Johansson. *Numerical Python Scientific Computing and Data Science. Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib*. Apress, 2nd editio edition, 2019.

[27] Analytics Vidhya. The Ultimate Guide to 12 Dimensionality Reduction Techniques (with Python codes).

[28] Laura Igual and Santi Seguí. *Introduction to Data Science. A Python Approach to Concepts, Techniques and Applications*. Springer International Publishing AG, 2017.

[29] Analytics Vidhya. Binary Cross Entropy aka Log Loss-The cost function used in Logistic Regression, 2020.

[30] Linux Team. LinuxMint, 2020.

[31] Python Software Foundation. Python 3.7.9 documentation, 2020.

[32] TIOBE Software. TIOBE Index, 2020.

[33] Inc Anaconda. Anaconda Documentation.

[34] The ObsPy Development Team (devs@obspy.org). ObsPy Tutorial. Release 1.1.1. 2019.

[35] Tobias Megies, Moritz Beyreuther, Robert Barsch, Lion Krischer, and Joachim Wassermann. ObsPy - what can it do for data centers and observatories? *Annals of Geophysics*, 54(1):47–58, 2011.

[36] Moritz Beyreuther, Robert Barsch, Lion Krischer, Tobias Megies, Yannik Behr, and Joachim Wassermann. ObsPy: A python toolbox for seismology. *Seismological Research Letters*, 81(3):530–533, 2010.

[37] Lion Krischer, Tobias Megies, Robert Barsch, Moritz Beyreuther, Thomas Lecocq, Corentin Caudron, and Joachim Wassermann. ObsPy: A bridge for seismology into the scientific Python ecosystem. *Computational Science and Discovery*, 8(1):1–17, 2015.

[38] IRIS Incorporated Research Institutions for Seismology. Data Formats IRIS, 2020.

[39] Román Lara-Cueva, Paúl Bernal, María Gabriela Saltos, Diego S. Benítez, and José Luis Rojo-Álvarez. Time and Frequency Feature Selection for Seismic Events from Cotopaxi Volcano. *Proceedings - 2015 Asia-Pacific Conference on Computer-Aided System Engineering, APCASE 2015*, pages 129–134, 2015.

[40] António E. Ruano, G. Madureira, Ozias Barros, Hamid R. Khosravani, Maria G. Ruano, and Pedro M. Ferreira. A support vector machine seismic detector for early-warning applications. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 3(PART 1):405–410, 2013.

[41] Román Lara-Cueva, Valeria Paillacho-Salazar, and Michelle Villalva-Chaluisa. Towards an automatic detection system of signals at cotopaxi volcano. *Dyna*, 84(200):176–184, 2017.

[42] A. E. Ruano, G. Madureira, O. Barros, H. R. Khosravani, M. G. Ruano, and P. M. Ferreira. Seismic detection using support vector machines. *Neurocomputing*, 135:273–283, 2014.

[43] Anvesh Parashar and Abhilash Sonker. Application of hyperparameter optimized deep learning neural network for classification of air quality data. *International Journal of Scientific and Technology Research*, 8(11):1435–1443, 2019.

[44] Mikko Kotila. Autonomio Talos, 2019.