Universidad Católica de Santa María

Facultad de Ciencias e Ingenierías Físicas y Formales

Escuela Profesional de Ingeniería de Minas

"COMPARACIÓN DE MODELOS EMPÍRICOS TRADICIONALES CONTRA MODELOS PREDICTIVOS DE MACHINE LEARNING PARA LA ESTIMACIÓN DE VELOCIDAD PICO PARTÍCULA EN VOLADURAS DE CIELO ABIERTO"

Tesis presentada por la Bachiller:
Valdivia Salazar, Roger Alonso
para optar el Título Profesional de
Ingeniero de Minas

Asesor:
Mg. Sulla Torres, José Alfredo

Arequipa- Perú

2023

UCSM-ERP

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

INGENIERIA DE MINAS

TITULACIÓN CON TESIS

DICTAMEN APROBACIÓN DE BORRADOR

Arequipa, 18 de Julio del 2022

Dictamen: 002872-C-EPIM-2022

Visto el borrador del expediente 002872, presentado por:

2015200741 - VALDIVIA SALAZAR ROGER ALONSO

Titulado:

COMPARACIÓN DE MODELOS EMPÍRICOS TRADICIONALES CONTRA MODELOS PREDICTIVOS DE MACHINE LEARNING PARA LA ESTIMACIÓN DE VELOCIDAD PICO PARTÍCULA EN VOLADURAS DE CIELO ABIERTO

Nuestro dictamen es:

APROBADO

1748 - CALDERON RUIZ GUILLERMO ENRIQUE
DICTAMINADOR

2872 - LOPEZ CASAPERALTA PATRICIA YANETH
DICTAMINADOR

3324 - DELGADO PONCE MARIA AZUCENA
DICTAMINADOR

Dedicatorias

Dedico este trabajo a mi madre, por ser motor y motivo en mi vida personal y profesional; por ser mi fuente de motivación y fuerza cada día; y por empujarme a dar lo mejor de mí en todo lo que haga.

A mi padre, por ser el más grande modelo de superación a seguir en mi vida; por enseñarme a discernir entre lo bueno y lo malo; y por siempre ofrecerme su apoyo en todas mis decisiones.

A mi hermana, por ser el mejor ejemplo de disciplina y esfuerzo, pero a la vez, de alegría y diversión; por quererme sin importar nada y por siempre querer verme feliz.

A ustedes, mi amada familia, por ser lo más importante en mi vida.

Roger Alonso Valdivia Salazar

Agradecimiento

Agradezco a mi familia, por el soporte y apoyo diario brindado en mi vida personal y profesional; y por todo el amor que me ofrecen día a día.

Agradezco a todos aquellos docentes que con su dedicación y vocación lograron que me enamorase de la Ingeniería de Minas. Gracias por sus conocimientos y exigencia, que hoy por hoy son parte fundamental de mi vida profesional.

Agradezco a todos mis amigos, porque sin ustedes y los momentos de distracción y diversión que hemos tenido, este trabajo no hubiese sido posible.

Y finalmente, agradezco a todas y cada una de las personas que de alguna u otra manera, han contribuido a la elaboración de este trabajo de tesis, brindándome apoyo, consejo y/o motivación.

RESUMEN

El presente estudio busca realizar una comparación objetiva y cuantitativa de modelos empíricos tradicionales contra modelos predictivos de Machine Learning para la estimación de Velocidad Pico Partícula en voladuras de operaciones mineras a cielo abierto. Para ello se seleccionó 5 formulaciones convencionales y 6 modelos predictivos desarrollados en base a algoritmos de Machine Learning; se desarrolló un script en lenguaje de programación Python para entrenar y optimizar cada uno de estos modelos con la finalidad de estimar que tan bien pueden predecir los valores esperados de Velocidad Pico Partícula; y finalmente compararlos en base a métricas de desempeño. Para el presente estudio se escogieron dos métricas de desempeño, siendo estas el Coeficiente de Determinación (R2) y la Raíz del Error Cuadrático Medio (RMSE). Tras la obtención y análisis de resultados, se concluyó que el modelo desarrollado en base a Redes Neuronales Artificiales fue el que mejores métricas obtuvo, con un valor de RMSE de 11,54 unidades y un valor de coeficiente de determinación (R2) igual a 0,88. Por otra parte, dentro de los modelos desarrollados mediante formulaciones empíricas, el que mejores métricas entregó fue el de Rai & Singh (2004); con un valor de RMSE de 20,65 unidades y un coeficiente de determinación (R2) de 0,61. En líneas generales, los modelos desarrollados en base a algoritmos de Machine Learning entregan mejores métricas de desempeño comparados respecto a los modelos desarrollados en base a formulaciones empíricas; representando así, una mejor opción en la tarea de predecir valores de Velocidad Pico Partícula.

Palabras claves:

Minería, Voladura, Velocidad Pico Partícula, Modelamiento Predictivo, Machine Learning.

ABSTRACT

The present study aims to perform an objective and quantitative comparison between traditional empirical models and Machine Learning predictive models for the estimation of Peak Particle Velocity in blasting in open pit mining operations. For this purpose, 5 conventional formulations and 6 predictive Machine Learning algorithms were selected. A Python script was developed to train and optimize each of these models to estimate how well they can predict the expected values of Peak Particle Velocity. Finally, all the models were compared based on performance metrics. Two performance metrics were chosen for this study: Coefficient of Determination (R2) and the Root Mean Square Error (RMSE). After the results were obtained and analyzed, it was seen that the model developed based on Artificial Neural Networks was the one that obtained the best metrics, with an RMSE value of 11.54 units and a determination coefficient (R2) value equal to 0.88. On the other hand, among the traditional models, the one with the best metrics was the model developed with the formulation proposed by Rai & Singh (2004); with an RMSE value of 20.65 units and a determination coefficient (R2) of 0.61. In general, the models developed based on Machine Learning algorithms provide better performance metrics compared to the models developed based on empirical formulations; thus, representing a better option in the task of predicting values of Peak Particle Velocity.

Keywords:

Mining, Blasting, Peak Particle Velocity, Predictive Modelling, Machine Learning**.**

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Uno de los principales retos que enfrentan las operaciones mineras a tajo abierto es, el poder estimar el impacto que tendrá una voladura sobre la estabilidad de los bancos colindantes y la integridad de las estructuras cercanas. Es por esto que resulta de suma importancia poder diseñar una voladura con los parámetros correctos para cumplir el objetivo fundamental de esta, fracturar el macizo rocoso para su posterior acarreo, pero sin causar algún tipo de daño sobre las estructuras o bancos cercanos. Uno de los principales factores a considerar en un diseño de voladura óptimo es la Velocidad Pico Partícula, índice que nos permite medir las vibraciones en el suelo inducidas por la voladura, y que funciona como indicador en el control de daños en proyectos de voladura.

Es un común denominador dentro de las empresas mineras, el utilizar formulaciones empíricas para la estimación de la Velocidad Pico Partícula. Sin embargo, estas formulaciones, algunas de ellas con 60 años de antigüedad, presentan muchas limitantes; como la reducida cantidad de variables que consideran para la estimación, o la dependencia a constantes propias del sitio donde se busca hacer la estimación.

En el presente estudio se busca poner a disposición, de la industria y la academia, distintas alternativas a las formulaciones empíricas de estimación de valores de Velocidad Pico Partícula. Para ello, se buscará probar que los modelos predictivos desarrollados con algoritmos de Machine Learning son capaces de entregar estimaciones mucho más precisas y robustas que las entregadas por los modelos empíricos.

La realización de este estudio cobra suma importancia debido al limitado conocimiento e implementación de estas técnicas en nuestro país, y a la creciente demanda de nuevas metodologías y tecnologías que puedan servir como alternativa de solución a los recurrentes problemas que se experimentan en la industria. Además, la realización de este estudio adquiere mayor relevancia debido a la inexistente publicación de estudios en nuestro país, e incluso, nuestro continente, que aborden la problemática mencionada con una metodología nueva, como la propuesta.

CAPÍTULO I

1. PLANTEAMIENTO DE LA INVESTIGACIÓN

1.1. Planteamiento del Problema

El lograr predecir y controlar de manera eficiente las vibraciones generadas por la voladura es considerado como uno de los mayores desafíos que enfrentan las empresas mineras. La generación desmedida e incontrolada de estas vibraciones significa un uso ineficiente de la energía liberada por los agentes explosivos durante la voladura, puesto que esta energía debería utilizarse, en el mayor grado posible, para el fracturamiento del macizo rocoso (Khandelwal, et al., 2017). Además de ello, las vibraciones traen consigo un número considerable de efectos adversos para la operación minera y el ambiente. No solo generan un grado de incomodidad a las poblaciones aledañas a la operación minera, sino que también pueden dañar de manera considerable las estructuras cercanas al proyecto de voladura. De igual manera, es importante mencionar que las vibraciones en el macizo rocoso pueden generar agrietamientos, o incluso fracturas, en los bancos colindantes a la voladura, lo que significará, la generación de situaciones de riesgos y no deseada en la operación, e incluso pueden llegar a dificultar las labores de perforación en futuros proyectos de voladura (Dehghani & Ataee-pour, 2011).

Todos estos efectos adversos, afectan las finanzas de la operación minera, es debido a esto que en la actualidad y desde hace ya bastantes décadas existe la necesidad de poder predecir el comportamiento de las ondas vibratorias producidas durante el fracturamiento del macizo rocoso producto de la voladura, con la finalidad de prever y minimizar los efectos negativos expuestos con anterioridad. Para lograr este objetivo la mayoría de las empresas mineras utilizan formulaciones empíricas propuestas hace ya bastantes décadas, como por ejemplo el modelo propuesto por Duvall (1962) o el propuesto por Langefors – Kihlstrom (1978). Estos modelos, si bien es cierto, han brindado resultados aceptables a lo largo de los años, presentan ciertas limitaciones, puesto que la totalidad estos ajustan el comportamiento de la velocidad pico partícula a un modelo exponencial en función de la distancia entre el punto de medición y la voladura, y la cantidad máxima de explosivo por detonador utilizada durante la voladura. A pesar de que, este criterio no es incorrecto, se limita de manera negativa al modelo predictivo, que muchas veces presenta un comportamiento mucho más complejo que el exponencial y que se ve afectado por muchos más factores, como por ejemplo las propiedades físico-mecánicas del macizo rocoso o las

propiedades del explosivo a utilizar. Todas estas condiciones tienen influencia considerable en la generación de las ondas vibratorias, y por ende deben ser consideradas en un modelamiento predictivo de la velocidad pico partícula, parámetro mediante el cual se representa de manera cuantitativa a las ondas vibratorias (Tosun, 2020).

En base a lo expuesto, surge la necesidad de presentar alternativas que puedan reemplazar a los modelos convencionales, y que, a su vez, logren reflejar de mejor manera el comportamiento de las ondas vibratorias en el macizo rocoso y que, en consecuencia, entreguen mejores resultados de correlación y de exactitud al momento de evaluar su desempeño. Todo esto con el objetivo de obtener mejores y más acertadas predicciones en los estudios de velocidad pico partícula que se realicen en las operaciones mineras a tajo abierto, y de esta manera poder diseñar las voladuras con parámetros más ajustados a la realidad, logrando disminuir los efectos negativos de este proceso.

1.2. Objetivos de la Investigación

1.2.1. General

Desarrollar una comparación objetiva y cuantitativa de los modelos empíricos tradicionales contra modelos predictivos de Machine Learning, para la estimación de velocidad pico partícula en voladuras de cielo abierto; con la finalidad de encontrar y presentar una alternativa de estimación más precisa a las metodologías utilizadas actualmente.

1.2.2. Específicos

A. Definir los conceptos generales que abarca el fracturamiento del macizo rocoso y la generación de ondas sísmicas, así como los aspectos teóricos detrás de los algoritmos a utilizar para el modelamiento predictivo.

B. Establecer un caso de estudio con datos reales que pueda servir como medio de comparación entre los modelos convencionales y los modelos predictivos propuestos.

C. Determinar métricas cuantitativas que evalúen los diferentes modelos, en base a que tan bien puede reflejar los datos de estudio y que tan bien puede predecir valores nuevos; y que sirvan como parámetro de

comparación entre los distintas modelos propuestos y los modelos convencionales.

D. Seleccionar que algoritmos de Machine Learning utilizaremos para desarrollar los diferentes modelos predictivos propuestos y desarrollar un programa o script en lenguaje de programación Python para cada uno de los modelos propuestos que determine los parámetros de configuración óptimos que necesita cada algoritmo para minimizar el error en las predicciones; y que finalmente, los califique en base a las métricas establecidas.

E. Determinar la importancia y el impacto en las predicciones de los parámetros de entrada considerados para el desarrollo de los modelos predictivos de Machine Learning.

F. Desarrollar un programa o script en lenguaje de programación Python que desarrolle la formulación de los modelos empíricos, determine las constantes del sitio óptimas que necesitan los modelos empíricos para minimizas el error en las predicciones, y los califique en base a las métricas establecidas.

G. Realizar una comparación objetiva de los resultados obtenidos en todos los modelos.

1.3. Pregunta de Investigación

¿Cuáles serán los resultados de desarrollar una comparación objetiva y cuantitativa de los modelos empíricos tradicionales contra modelos predictivos de Machine Learning, para la estimación de velocidad pico partícula en voladuras de cielo abierto?

1.4. Campo, Área y Línea de Investigación

1.4.1. Campo: Ingeniería

1.4.2. Área: Ingeniería de Minas

1.4.3. Línea: Optimización de Proceso Minero

1.5. Palabras Clave

Minería, Voladura, Velocidad Pico Partícula, Modelamiento Predictivo, Machine Learning, Regresión Lineal Múltiple, Árboles de Decisión, Bosques Aleatorios, Potenciación de Gradiente, Máquina de Vectores de Soporte, Redes Neuronales Artificiales.

Mining, Blasting, Peak Particle Velocity, Predictive Modelling, Machine Learning, Multiple Linear Regression, Decision Trees, Random Forest, Gradient Boosting Machine, *Support Vector Machine, Artificial Neural Networks.*

1.6. Solución Propuesta

1.6.1. Justificación e Importancia

Desde el punto de vista de la conveniencia o utilidad, la presente investigación cobra relevancia al presentar alternativas tentativas que permitan obtener un mejor modelo predictivo de la velocidad pico partícula, que permitiría a su vez, prever y controlar posibles daños en un campo lejano producto de voladuras en operaciones mineras a tajo abierto.

Desde el punto de vista social, a través del presente trabajo se busca beneficiar a aquellas empresas mineras que requieran una alternativa mucho más precisa a los modelos convencionales de velocidad pico partícula en campo lejano, logrando diseños de voladuras mejor controlados, evitando así posibles daños a estructuras aledañas o incluso a la estabilidad del macizo rocoso; resultando indirectamente en un beneficio económico para la empresa minera.

Desde el punto de vista teórico, la presente investigación aportará mayores conocimientos, datos relevantes, y sentará las bases para futuras investigaciones que decidan ahondar en el tema de la inteligencia artificial aplicada a crear modelos predictivos en minería. Se espera determinar la precisión de los modelos propuestos y demostrar su aplicabilidad en el campo minero.

Finalmente, yo como estudiante egresado de la escuela profesional de Ingeniería de minas, llevo a cabo la presente investigación con la finalidad de optar por el título profesional de Ingeniero de Minas.

1.6.2. Descripción de la Solución

Con la realización de la presenta investigación se espera presentar a la industria minera, mejores modelos predictivos como alternativas a los modelos convencionales, y que estos modelos logren reflejar de mejor manera el comportamiento de la velocidad pico partícula en el campo de estudio y que, en consecuencia, entreguen mejores resultados de correlación y de exactitud al momento de evaluar su desempeño. Se espera presentar satisfactoriamente una solución a la problemática expuesta, y no se esperan inconvenientes a lo largo del desarrollo de la presente investigación

1.7. Aporte

Los algoritmos de aprendizaje autónomo o basados en Machine Learning apuntan a ser parte fundamental de la solución u optimización de problemas en un futuro cercano, es por esto por lo que, el presente proyecto de investigación servirá como antecedente para futuros investigadores que deseen ahondar en el tema de la implementación de modelos computacionales en la industria minera, especialmente para la optimización o actualización de procesos. Esto debido a la limitada o casi nula literatura existente en el idioma castellano generada en América Latina sobre este tema.

Además de su aporte científico o académico, la presente investigación brindará una solución integral a disposición de todas las empresas mineras que busquen una alternativa a las formulaciones empíricas de predicción para valores de Velocidad Pico Partícula con la finalidad de obtener resultados más acertados y de esta manera lograr minimizar los efectos adversos de las vibraciones generadas durante la voladura. Este aporte cobra considerable importancia debido al limitado conocimiento e implementación de estas técnicas en nuestro país, y a la creciente demanda de nuevas técnicas y tecnologías que permitan presentar alternativas de solución a los recurrentes problemas que se experimentan en la industria.

CAPÍTULO II

2. FUNDAMENTOS TEÓRICOS

2.1. Estado del Arte

En los últimos años se han desarrollado incontables investigaciones y estudios que intentan utilizar modelos convencionales de predicción de velocidad pico partícula para optimizar o bien, controlar el proceso de voladura en distintas unidades mineras de alrededor del mundo, pero es muy poca la información existente sobre alternativas a los métodos de predicción convencionales como las que se presentan en la investigación planteada. A continuación, se presentará los estudios que han podido recopilarse en bases de datos especializadas y revistas indexadas, esto con la finalidad de sentar las bases y presentar los antecedentes de esta investigación.

Una investigación que evalúa alternativas al modelo predictivos convencional presentando algoritmos de Machine Learning es la presentada por Manoj Khandelwal, Danial Jahed Armaghani, Roohollah Shirani Faradonbeh, Mohan Yellishetty, Muhd Zaimi Abd Majid y Masoud Monjezi, titulado ***"Classification and regression tree technique in estimating peakparticle velocity caused by blasting",*** estudio publicado en Londres, Inglaterra en el año 2016 y que se encuentra disponible de manera virtual en: https://doi.org/10.1007/s00366-016-0455-0. Y que busca encontrar y presentar una solución siguiente problema: Las vibraciones producidas por la voladura son consideradas como el fenómeno menos deseado en una operación minera, pueden producir daños en las estructuras cercanas, daños al macizo rocoso, a las rampas, a las operaciones subterráneas, al talud, a las aguas subterráneas y puede dañar irreparablemente la ecología de las zonas circundantes. Árboles de Regresión y Clasificación (CART, por sus siglas en inglés) fueron utilizados para predecir valores de PPV a través de una base de datos de 51 datos, que incluyen la carga máxima por retardo y la distancia del punto de medición a la voladura. Adicionalmente se utilizaron, un modelo empírico propuesto por la United States Bureau of Mines (USBM), y un modelo de regresión múltiple, esto para poder comparar entre un número mayor de alternativas utilizadas actualmente. Y finalmente. el desempeño de los modelos propuestos fue comparados y evaluados usando 3 criterios estadísticos: Coeficiente de correlación (R2), Raíz del Error Cuadrático Medio (RMSE, por sus siglas en inglés) y Contabilidad de la Varianza (VAF, por sus siglas en inglés). Teniendo en cuenta

los conjuntos de datos de prueba, se obtuvieron los siguientes resultados; valores de R2 (0.92, 0.89 y 0.87), VAF (91.89, 86.94 y 86,94), y RMSE (0,97, 0,99 y 0,98) para los modelos CART, empírico y de regresión múltiple, respectivamente. El modelo CART demuestra superioridad y más confiabilidad en la tarea de predecir velocidad pico partícula, comparado con el modelo de regresión lineal y los modelos empíricos; lo que demuestra que puede ser introducido en la industria minera como una nueva técnica. (Khandelwal, et al., 2017)

Un segundo estudio que de igual manera al presentado previamente y a la investigación planteada en esta tesis, busca presentar alternativas para la predicción de valores de VPP utilizando herramientas de Inteligencia Artificial es el estudio titulado: ***"Design of blasting pattern in proportion to the peak particle velocity (PPV): Artificial Neural Network Aprroach",*** presentado por H. Bakhshandeh Amnieh, A. Siamaki y S. Soltani en Irán en el año 2012, disponible de manera virtual en: https://doi.org/10.1016/j.ssci.2012.05.008. Este estudio presenta como problema que las vibraciones en el suelo son producidas en las voladuras, resultando en daños a edificaciones y otros bienes cercanos a los frentes de minado. Y resalta el hecho que estas mismas vibraciones en el suelo son afectadas y "controladas" por distintos factores de diseño de voladura: distancia al frente de minado, masa de explosivo utilizada, etc. Utilizando data recolectada de 20 voladuras en 47 puntos distintos, y otros parámetros como volumen del bloque a volar o tipo de explosivo utilizado, se busca dar solución a la problemática entrenando una Red Neuronal Artificial (ANN, por sus siglas en inglés) para que pueda predecir los parámetros de diseño necesarios en la voladura para no causar daños en las estructuras cercanas a la voladura. Exactamente, el modelo en base a la ANN nos entrega como valores de salida o resultados, valores de burdem, espaciamiento y carga total por voladura, significando un acortamiento de tiempo y recursos, puesto que ya no será necesario rediseñar o calcular valores exactos para estos parámetros de diseño. Se midió la efectividad del modelo utilizando el coeficiente de correlación para cada uno de los parámetros de salida del modelo, obteniendo los siguientes resultados: 0.651 para la masa total de explosivo, 0.77 para el burdem y 0.963 para el espaciamiento. La mitigación de las vibraciones causadas en la voladura, disminuyen no solo el daño a las estructuras cercanas

a la operación minera sino también el grado de insatisfacción de los poblados cercanos a la minera, por lo tanto, resulta importante presentar alternativas de predicción que entreguen datos mucho más acertados y reales que los obtenidos usualmente por métodos convencionales. (Bakhshandeh Amnieh, Siamaki, & Soltani, 2012)

Un tercer estudio que ahonda los temas mencionados hasta este momento es el presentado por H. Dehghani y M. Ataee-pour, titulado ***"Development of a model to predict peak particle velocity in a blasting operation",*** investigación publicada en Irán durante el año 2012, disponle de manera virtual en: https://doi.org/10.1016/j.ijrmms.2010.08.005 y que presenta la siguiente problemática: Las vibraciones en el suelo provenientes de voladuras en macizos rocosos son uno de los problemas fundamentales en la industria minera. Estas vibraciones consumen energía de la voladura que podría haber sido utilizada en el fracturamiento de la roca. La intensidad de las vibraciones desempeña un papel importante en todo tipo de efectos adversos; no solo genera problemas e incomodidad en poblaciones aledañas a las operaciones, sino también afecta negativamente a las estructuras de la operación minera. También daña las aguas subterráneas y afecta la ecología de la zona. Estas mismas vibraciones pueden producir daños en la cara libre y grietas en los bancos, que a su vez producirían dificultades en la perforación del siguiente proyecto de voladura, resultando en voladuras deficientes con bolonería de gran tamaño. Todos esto, afecta las finanzas de la empresa minera, la producción y pone en peligro el desarrollo socioeconómico de las localidades aledañas. Entonces, resulta importante medir, controlar y predecir con alta precisión las vibraciones. Para lograr esto, varios parámetros como: las propiedades físico-mecánicas del macizo rocoso, características del explosivo, parámetros geométricos de diseño y consideraciones de los tiempos de retardo, deberían considerarse en el diseño correcto de una voladura. En el pasado, los modelos convencionales eran utilizados para diseñar los parámetros de voladura. Pero debido a la gran cantidad de parámetros y a la complejidad de la relación entre los mismos, los métodos de diseño convencionales no son del todo adecuados. Para evaluar y calcular las vibraciones producidas por voladuras, incorporando parámetros de diseño de voladura y factores del macizo rocoso, se utilizará una red neuronal artificial y técnicas de análisis dimensional. Primeramente, se utilizará una red neuronal de tres capas, con 9 parámetros de entrada en la

primera capa, 25 unidades escondidas en la segunda capa y una sola unidad de salida (VPP) en la última capa; esta red neuronal será utilizada para entrenar 116 mediciones de voladuras provenientes de una de las minas de cobre más importantes en Irán. 17 nuevos ejemplos serán utilizados para la validación del modelo. En la segunda etapa se desarrollará una nueva fórmula utilizando análisis dimensional. Finalmente, se comparará los modelos convencionales con la nueva fórmula, utilizando el Coeficiente de Correlación (R2) y la Raíz del Error Cuadrático Medio (RMSE) entre los valores de PPV medidos y los predichos para todos los modelos. Además de proveer las mejores predicciones de las vibraciones, la nueva fórmula obtuvo los mejores puntajes en coeficiente de correlación y RMSE, 75.5% y 3.49, respectivamente. Con la base de los resultados obtenidos, se concluye en el presente estudio que ANN es una consistente y versátil técnica para optimizar la eficiencia de la voladura en operaciones mineras a cielo abierto, permitiéndonos controlar un fenómeno indeseable. El algoritmo de propagación inversa demostró ser el más eficaz para el aprendizaje del modelo. Se encontró que la arquitectura de la red neuronal (9-25-1), es la óptima para predecir valores de velocidad pico partícula en la operación minera Sarcheshmeh en Irán. Análisis dimensional puede ser considerado como una importante herramienta para solucionar la mayoría de los problemas científicos. En este caso, el efecto en conjunto de distintos parámetros que afectan en la voladura puede ser estudiado con Análisis Dimensional. Y tras la evaluación de los resultados se pudo concluir que las vibraciones se ven afectadas en gran parte por factores como el factor de carga, carga por retardo y el burdem. (Dehghani & Ataee-pour, 2011)

El siguiente estudio que se tomará en cuenta en la presente investigación como base o antecedente es el presentado y publicado en Alemania durante el año 2019, titulado ***"Predicting blast-induced peak particle velocity using BGAMs, ANN and SVM: a case study at the Nui Beo open-pit coal mine in Vietnam"*** (disponible de manera virtual en: https://doi.org/10.1007/s12665-019-8491-x**,** cuyos autores son: Hoang Nguyen, Xuan-Nam Bui, Quang-Hieu Tran y Hossein Moayedi. A continuación, se presenta un resumen de la investigación, considerando problemática, alternativa de solución, resultados y conclusiones. Uno de los efectos más perjudiciales que encontramos durante las voladuras en operaciones mineras a cielo abierto, son las vibraciones en el suelo. La velocidad pico

partícula (PPV) es una métrica comúnmente utilizada para medir las vibraciones en el suelo, pero, obtener predicciones precisas y acertadas de valores de PPV, resulta desafiante para los ingenieros de voladura y para los gerentes de las empresas. Aproximadamente, 25-30% de la energía explosiva liberada en la voladura se utiliza eficazmente para fracturar la roca, el resto de la energía tienes efectos indeseables para el ambiente, como vibraciones, agrietamientos, fragmentos de roca proyectados, generación de polvo y liberación de componentes tóxicos. Dentro de todos estos efectos, las vibraciones, definido en términos de VPP, es el efecto secundario considerado más severo, que puede llevar a consecuencias como: inestabilidad en los taludes y bancos, daños en estructuras, causar fracturamiento o incluso derrumbes, afectar rampas de acceso, vías de tren y reservorios de aguas subterránea. Muchas empresas mineras han enfrentado problemas legales debido a los efectos de las vibraciones generadas por la voladura, llegando en algunos casos a significar el cierre permanente o temporal de sus operaciones. Es por todo lo expuesto, que las vibraciones producidas en la voladura (PPV) necesitan ser predichas de manera acertada para poder controlar sus efectos adversos en el ambiente colindante. Muchas ecuaciones empíricas han sido propuestas a lo largo de los años para poder predecir valores de PPV acertadamente, mencionando solo algunos, tenemos el modelo empírico propuesto por la USBM (1958), Langefors y Kihlstrom (1963), Roy (1991) and Wahyudi (2015). A pesar de todos estos intentos, los resultados obtenidos por los mismos fueron encontrados como insuficientes y no confiables. Es por ello que nuevas propuestas son necesarias, ya que la precisión es necesaria en este problema. Se utilizó un Modelo conocido como Boosted Generalized Additive Model (BGAM) para estimar los valores de PPV producidos en voladura, adicionalmente se utilizó una ecuación empírica, un modelo en base a una Máquina de Vectores de Soporte (SVM), y una Red Neuronal Artificial (ANN), para predecir valores y usarlos con fines de comparación. Se utilizó como ejemplo base una base de datos de 79 voladuras de la mina de carbón a cielo abierto Nui Beo, ubicada en Vietnam. 3 indicadores de desempeño fueron utilizados para evaluar la calidad de cada modelo predictivo, coeficiente de determinación (R2), raíz del error medio cuadrado (RMSE) y el error medio absoluto (MAE). De acuerdo con los resultados, el modelo aditivo generalizado (BGAM) presentó el mejor desempeño comparado al resto de modelos, presentando la mejor exactitud con un coeficiente de determinación R2 de 0.99, un RMSE

de 0.582 y MAE de 0.43. La red neuronal y el modelo SVM mostraron desempeños diminutamente inferiores, mientras que la formulación empírica, presento el peor desempeño. Dos voladuras de prueba fueron realizadas para validar la exactitud de los modelos desarrollados en un ambiente real, como era de esperarse, el modelo BGAM presento el mejor desempeño. Los resultados también apuntan que la diferencia en cuanto a elevación entre la voladura y el punto de monitoreo es uno de los parámetros predominantes cuando de predecir valores de PPV se trata. La Inteligencia Artificial (IA) nos da técnicas avanzadas que pueden reemplazar por completo las técnicas empíricas para predecir valores de PPV en voladuras. A pesar de que las técnicas en base a IA nos ofrecen mejores resultados, estas necesitan conocimientos de machine Learning y no son tan sencillos de implementar como una ecuación empírica. (Nguyen, Bui, Tran, & Moayedi, 2019)

Un quinto estudio a considerar por su relevancia y similitud en metodología con la investigación planteada en esta tesis, sería el estudio desarrollado y publicado por Taha Mokfi, Azam Shahnazar, Iman Bakhshayeshi, Ali Mahmodi Derakhsh y Omid Tabrizi, titulado ***"Proposing of a new soft computing-based model to predict peak particle velocity induced by blasting"*** con lugar y año de publicación: Londres, Inglaterra – 2018 y disponible de manera virtual en: https://doi.org/10.1007/s00366-018-0578-6 Presenta un enfoque muy parecido al planteado en la presente investigación y es por ello que a continuación se presenta un resumen el mismo. La estimación de las vibraciones del suelo inducidas por las operaciones de voladura es una tarea importante para controlar los problemas de seguridad en operaciones mineras superficiales y proyectos civiles. Revisando estudios previamente realizados, se encontró que algunos modelos empíricos han sido propuestos, al igual que algunos modelos "Soft-Computing", con el fin de poder estimar vibraciones producidas por la voladura. La perforación y voladura es una técnica muy conocida y económica para fragmentar macizos rocosos tanto en trabajos civiles como en operaciones mineras superficiales. Solo 20%-30% de la energía explosiva se aplica eficientemente a la fragmentación de la roca y el resto de la energía liberada en la voladura se disipa a lo largo del suelo desencadenando distintos efectos no deseados, tales como humos, levantamiento de polvo, ondas expansivas, fragmentos de roca expulsados y

vibración en el suelo. Dentro de todos estos efectos negativos, resalta la vibración de suelos, ya que puede causar daño a las estructuras colindantes, como taludes, rieles de tren, etc. Entonces, predecir de manera acertada las vibraciones, resulta de vital importancia para reducir o minimizar los efectos ambientales producidos en la voladura. El modelo denominado "Group Method of Data Handling" (GMDH), que es un tipo de red neuronal artificial, es propuesto para predecir valores de PPV en esta investigación. Utilizando como parámetros de entrada: largo del stemming, factor de potencia, relación del burdem con el espaciamiento, distancia desde la cara de voladura, profundidad de los taladros, y la carga máxima por retardo. La base de datos necesaria para realizar el modelo fue colectada de una cantera en Penang – Malasia, la base de datos consta de información recolectada de 102 voladuras. La base de datos se separó con una relación 80% - 20% para la data de entrenamiento y la data de prueba, respectivamente. Distintos criterios de evaluación fueron utilizados para medir la confiabilidad del modelo, tales como la raíz del error medio cuadrado (RMSE) y el coeficiente de determinación (R2). Adicionalmente al modelo ya mencionado, para fines de comparación se desarrollará y presentará resultados de los siguientes modelos: Gene Expression Programming (GEP) y Non-Linear Multiple Regression (NLMR). El modelo GMDH presentó un valor de RMSE de 0.889, y un coeficiente de determinación R2 de 0.911, valores superiores comparados con los obtenidos por el modelo GEP, que presentó valores de RMSE: 0.963 y R2: 0.874, y por el modelo NLMR, que presento resultados de RMSE: 1.498 y de R2: 0.79. Se concluye que el modelo GMDH puede ser considerado como una herramienta potente para predecir valores de Velocidad pico partícula, presentando valores superiores a los modelos empíricos utilizados comúnmente y a otros modelos computacionales. (Mokfi, Shahnazar, Bakhshayeshi, & Mahmodi, 2018)

Un estudio que plantea una alternativa a los modelos convencionales, sin ahondar en temas muy complejos de modelamiento computacional, pero que al igual que esta investigación tiene como objeto final, optimizar la predicción de valores de Velocidad Pico Partícula, es el estudio publicado y desarrollado por A. Tosun en Turquía en el año 2020, titulado ***"Modified Scale Distance Equation used for estimation of peak particle velocity"*** que se encuentra disponible de manera virtual en:

https://doi.org/10.1134/S1062739120036677. A continuación, se describirá la problemática expuesta, junto a la solución propuesta y a los resultados y conclusiones. Algunos efectos adversos al medio ambiente ocurren como resultado de los procesos de voladura. El efecto más considerable son las vibraciones en el suelo que pueden llegar a causar daños a edificaciones debido a sus altos niveles de energía. Estos niveles de energía pueden llegar a ser medidos dependiendo de un numero de parámetros a considerar, como, por ejemplo, desplazamientos de macizo rocoso, velocidad de vibración, aceleración de vibración y frecuencia de onda. Resulta importante realizar estimaciones acertadas de los valores de Velocidad Pico partícula antes de realizar las voladuras, para asegurar resultados adecuados en el menor daño posible al ambiente. Ecuaciones en base a la distancia son usadas usualmente para la estimación de valores de VPP, y distintas formulaciones se han desarrollado a lo largo de los años. Una de las ecuaciones más usadas y aceptadas es la presentada por Duvall y Fogelson. Sin embargo, esta ecuación no logra estimar de manera correcta los valores de VPP. En este estudio, los valores de velocidad pico partícula se midieron mediante una serie de pruebas de voladura llevadas a cabo en cuatro distintas canteras de caliza ubicadas en Turquía (Orhaneli, Çan, Bornova 1 y Bornova 2) y mediante el uso de un medidor de vibraciones, cabe resaltar que cada una de estas canteras presenta características de macizo rocoso únicas y completamente independientes. Los valores de distancia escalados para cada prueba de explosión se calcularon de acuerdo con ecuación propuesta por Duvall y Fogelson. Posteriormente, una serie de modificaciones fue propuesta considerando distinto rangos para la relación entre la distancia horizontal y la vertical medida entre el sitio de medición y la voladura en sí. Con los datos obtenidos, se realizó un análisis de correlación entre los valores de distancia escalar calculados por ambos métodos y los valores de VPP. En el primer sitio de estudio, utilizando la ecuación propuesta, el coeficiente de determinación R2 aumentó de 0,7861 a 0,8729 en comparación con la fórmula de Duvall y Fogelson, en el segundo sitio de estudio, desde 0,6648 a 0,932, en el tercer sitio de estudio, de 0,7705 a 0,8532. En el cuarto sitio de estudio, el coeficiente de la determinación aumentó de 0,8090 a 0,9272. Las pruebas de voladura realizadas en los cuatro diferentes sitios de estudio con diferentes características del macizo rocos mostraron que la relación propuesta que incluye los componentes vertical y horizontal garantiza una

mayor correlación entre la velocidad máxima de las partículas y la distancia escalar, en comparación con la ecuación propuesta por Duvall y Fogelson. (Tosun, 2020)

Otro estudio para tomar en cuenta considerablemente es el presentado por A.E. Alvarez-Vigil, C. Gonzalez-Nicieza, F. Lopez Gayarre y M. I. Alvarez-Fernandez, titulado ***"Predicting blasting propagation velocity and vibration frequency using artificial neural networks"*** publicado en España en el 2011 y que se puede encontrar de manera virtual en el siguiente enlace: https://doi.org/10.1016/j.ijrmms.2012.05.002. Por presentar un enfoque muy similar al que se evaluará en la presente investigación, a continuación, se presenta un resumen del mismo. El crecimiento de la industria minera debido a la creciente demanda de minerales ha llevado a un incremento considerable en el uso de explosivos para propósitos de voladura. Los explosivos son una eficiente fuente de energía para romper y quebrar macizo rocoso. Un explosivo detonado dentro de un taladro de perforación libera una cantidad masiva de energía en forma de presión y temperatura. A pesar de que constantemente de desarrollan y presentan nuevos e importantes avances tecnológicos en el área de la voladura, no se han alcanzado logros significativos debido a la complejidad de los parámetros que caracterizan al macizo rocoso a ser fragmentado. Cabe resaltar, que solamente una pequeña porción de la energía explosiva es usada para fragmentar el macizo rocoso, la mayor parte de la energía se disipa generando efetos no deseados como vibraciones, ruido, polvo, fragmentos expulsados, etc. El efecto "menos" deseado por las operaciones mineras son las vibraciones, por el efecto negativo y perjudicial que provoca en todo el ambiente alrededor de la voladura. La frecuencia (F) y la velocidad pico partícula (VPP) son dos de los parámetros más utilizados para medir las vibraciones en el suelo. Es importante entender el efecto de estos parámetros para poder usar la energía de manera efectiva y minimizar efectos no deseados en la voladura. Parámetros de diseño; tales como la carga explosiva por retardo, tiempos de retardo, espaciamiento, burdem, secuencia de salida; varían considerablemente de mina a mina, incluso puede variar considerablemente dentro de la misma operación minera. De igual manera, las características del macizo rocoso pueden variar en cortas distancias, es por ello que los parámetros de diseño de voladura y las características del explosivo a utilizar necesitan ser optimizados considerando las características del macizo rocoso. Se utilizará una red neuronal artificial

con el objetivo de predecir valores máximos de VPP y frecuencia de las vibraciones en el suelo a partir de la información sobre las propiedades físicas y mecánicas del macizo rocoso, de las características del explosivo y de los parámetros de diseño de la voladura. Se consideraron los parámetros que posiblemente podrían tener influencia en la predicción. Se hizo una distinción entre dos tipos de parámetros: aquellos que definen el entorno en el que se propaga la onda (tipo de roca, macizo rocoso, alcance posible de la onda y las discontinuidades significativas del subsuelo) y aquellos que definen la energía de la onda (el tipo de explosivo, cantidad de explosivo, parámetros geométricos de la voladura y secuencia de salida). Las vibraciones fueron monitoreadas usando sismógrafos capaces de capturar datos de vibración y transformarlos a términos de aceleración y frecuencia de onda. Para validar esta metodología, los resultados obtenidos por la predicción fueron comparados con los obtenidos mediante métodos estadísticos convencionales, usando métricas como el error medio cuadrado y el coeficiente de correlación. El coeficiente de correlación obtenido por la metodología descrita fue de 0.98 para los valores de Velocidad Pico Partícula y de 0.95 para los valores de Frecuencia, valores sumamente superiores comparados con los valores obtenidos por el método convencional de regresión Lineal múltiple: 0.50 y 0.15. Considerando la complejidad en la relación entre los datos de entrada y los datos de salida de la red neuronal, los resultados obtenidos son sumamente satisfactorios y prometedores. La red Neuronal es capaz de predecir valores con nuevos datos de entrada. Esto garantiza un alto nivel de certeza para predicciones futuras. Utilizando esta metodología, el monitoreo durante las voladuras puede ser eliminado por completo, resultando en un ahorro sustancial en mano de obra y materiales. (Álvarez-Vigil, González-Nicieza, López Gayarre, & Álvarez-Fernández, 2012)

Otro estudio para tomar en cuenta es el titulado ***"A new hybrid ANFIS–PSO model for prediction of peak particle velocity due to bench blasting"*** (disponible de manera virtual en: https://doi.org/10.1007/s00366-016-0438-1), esto debido a que la problemática que se busca solucionar es la misma que la presentada en esta investigación, y el enfoque utilizado para buscar una solución es muy parecido al utilizado en este estudio. Este estudio preparado, desarrollado y publicado por Ebrahim Ghasemi, Hamid Kalhori y Raheb Bagherpour, en la ciudad de Londres, Inglaterra, durante el año 2016, presenta al lector la

problemática que puede llegar a significar las vibraciones en el suelo. Se menciona que a pesar de que las voladuras son una de las operaciones elementales más importantes en el proceso minero, siempre vendrá acompañada de efectos secundarios no deseados; las vibraciones en el suelo son uno de las más, sino el más, destructivo de estos efectos secundarios y que lleva aquejando a los ingenieros por mucho tiempo. También se describe de manera concisa como es que se forman estas ondas expansivas que se reflejan en vibraciones; se menciona que cuando un explosivo detona dentro de un taladro perforado, esfuerzos dinámicos de alta intensidad se distribuyen alrededor del taladro debido a la repentina aceleración del macizo rocoso por la expansión de gases y la presión que estos ejercen sobre el mismo. Las ondas de tensión transmitidas a la roca colindante generan un movimiento ondulatorio en el suelo. La energía tensional llevada por estas ondas tensionales fragmenta el macizo rocoso debido a distintos mecanismos de rotura como, por ejemplo, trituración, fracturas radiales o rotura de reflejo en la presencia de una cara libre. Cuando la intensidad de la onda de esfuerzos decae al nivel donde dejan de ocurrir deformaciones en el macizo, las ondas de esfuerzo se propagan a través del medio, como una onda elástica, oscilando las partículas por las que viajan. Estas ondas en la zona elástica son conocidas comúnmente como vibraciones en el suelo. Vibraciones de alto nivel provocan daños negativos y dañinos en las estructuras cercanas, el agua subterránea y el ecosistema de la zona cercana. Se menciona que, para minimizar estos efectos adversos y dañinos, los ingenieros de minas y de voladura deben monitorear y supervisar el fenómeno de vibraciones con mucho cuidado, y que la VPP es el indicador más práctico para evaluar este fenómeno. Un último hecho considerado por los autores y que se resalta en la problemática propuesta es el hecho que a lo largo de los últimos años, un sinnúmero de ecuaciones empíricas han sido propuestas para predecir valores de VPP, siendo la más usada, la “Escalar Raíz Cuadrada” presentada por The United States Bureau of Mines (USBM). Se hace importante mención en el texto sobre las grandes desventajas que trae consigo seguir utilizando una formula empírica, como, por ejemplo, que estas formular son específicas para cada mina y no se pueden reutilizar en otros lugares, y que solo consideran dos parámetros para predecir VPP, cantidad máxima de explosivo por retardo y distancia a la voladura. Más adelante en este estudio se presenta el tema de las técnicas de computación suave como alternativa de solución a la poca eficacia y asertividad de las

fórmulas empíricas. Y se indica que, en el estudio a mención, se busca predecir valores de VPP utilizando dos técnicas de computación suave: hybrid adaptive neuro-fuzzy inference system and particle swarm optimization (ANFIS–PSO) y support vector regression (SVR). Para los propósitos e investigación, se utilizó una base de datos recopilada en la mina de cobre Sarchesmeh, considerando parámetros de diseño de la voladura. De acuerdo a los resultados obtenidos, se concluye que ambas técnicas pueden ser utilizadas para predecir los valores de VPP, pero que, tras la comparación de los modelos, el modelo ANFIS-PSO entrega mejores resultados, un valor de raíz del error cuadrado medio (RMSE por sus siglas en inglés) igual a 1.83, un coeficiente de determinación (R2) igual a 0.97, mientras que para el modelo SVR, se obtuvieron valores de RMSE igual a 2.4 y de coeficiente de determinación igual a 0.924. (Ghasemi, Kalhori, & Bagherpour, 2016)

Un estudio a considerar como base o antecedente sería el publicado en China (2020) por Jiandong Huanhg, Mohammadreza Koppialipoor y Daniel Jahed Armaghani, titulado ***"A combination of fuzzy Delphi method and hybrid ANN-based systems to forecast ground vibration resulting from blasting"*** (disponible de manera virtual en: https://doi.org/10.1038/s41598-020-76569-2), debido a su reciente publicación y por la magnitud del enfoque que presenta para comparar un gran número de modelos predictivos con la finalidad de tener una carta mucho más amplia de alternitas de donde elegir al modelo que entregue mejores resultados. A continuación, se presenta un resumen de este estudio, considerando una descripción de la problemática, el enfoque de solución y finalmente los resultados obtenidos acompañador de una breve conclusión. Hoy en día, el proceso de voladura es reconocido como el método con mejores resultados y equilibrio en términos de costo, en cuanto a fragmentación de macizo rocoso se refiere, tanto en construcción de túneles, operaciones mineras a cielo abierto u obras civiles. El principal propósito de la voladura es lograr un fracturamiento optimo del macizo rocoso para poder hacer más fácil el proceso de carga y transporte. En el otro lado de la moneda, a pesar de ser muchos los beneficios que trae la voladura, también traen consigo algunos resultados no deseados y perjudiciales para el ambiente, como las vibraciones en el suelo, fragmentos de roca eyectados u ondas expansivas en el aire. Es por ello, que existe una creciente necesidad por diseñar modelos que puedan evaluar y predecir de manera efectiva y acertada

estos fenómenos no deseados, lo que significaría aumentar de manera significativa los niveles de seguridad en las operaciones de voladura. Dentro de todos los efectos no deseados mencionados, las vibraciones en el suelo producidas por la voladura son considerados por muchos ingenieros como los efectos más perjudiciales, debido a los efectos que puede traer, como daños en estructuras cercanas, inestabilidad en taludes y causar molestia en comunidades aledañas. Por lo ya mencionado, es de vital importancia e intereses desarrollar una técnica de precisión para predecir vibraciones en el suelo producidas por la voladura, que nos permita controlar y minimizar este parámetro antes de la voladura. La velocidad pico partícula (VPP) puede indicar significativamente el control de los posibles daños que causan las vibraciones. Dos parámetros cobran vital significancia a la hora de evaluar valores de VPP, la carga máxima por retardo y la distancia a la cara libre. Muchos investigadores han estudiado el fenómeno de la VPP, resultando la gran mayoría de estos estudios en dos limitantes: el bajo nivel de precisión que se obtiene, y son muy específicos, esto quiere decir que solo se diseñan para un determinado lugar y no pueden ser replicados en otra mina. De igual manera, algunas técnicas estadísticas han sido utilizadas, pero estas se ven limitadas por el número de parámetros que se debería considerar a la hora de diseñar un modelo predictivo. Para superar todas las limitantes que presentan los modelos empíricos y estadísticos, es necesario utilizar nuevos sistemas que posean la habilidad de solucionar problemas más complejos. Modelos basados en Inteligencia artificial (IA) son capaces de brindar una solución óptima estos complejos problemas. Las redes neuronales artificiales pueden utilizarse para predecir valores de alta complejidad, pero a su vez sufren de algunas limitaciones, pueden caer en mínimos locales y no encontrar la solución óptima, o ser muy lentos a la hora de entrenarse. Sin embargo, se pueden utilizar algunos algoritmos de optimización para superar estas limitaciones, como, por ejemplo: Imperialism Competitive Algorithm (ICA), Genetic Algorithm (GA), Artificial Bee Colony (ABC), Firefly Algorithm (FA), and Particle Swarm Optimization (PSO). Con la finalidad de predecir de manera efectiva valores de VPP, el presente estudio desarrollara 5 redes neuronales hibridas, utilizando los algoritmos de optimización ya mencionados. Se considerarán los parámetros de entrada que son considerados clave por expertos en el área de voladura. Mediante un sistema de ranking simple, se seleccionó el mejor modelo híbrido. Los resultados obtenidos revelaron que el modelo FA-ANN es capaz

de ofrecer mayor nivel de precisión para la predicción de PPV en comparación con los otros modelos híbridos implementados. Se obtuvieron valores de coeficiente de determinación (R2) de (0.8831, 0.8995, 0.9043, 0.9095 y 0.9133) y (0.8657, 0.8749, 0.8850, 0.9094 y 0.9097) para las etapas de entrenamiento y prueba de los modelos GA-ANN, PSO-ANN, ICA-ANN, ABCANN y FA-ANN, respectivamente. Los resultados mostraron que todos los modelos híbridos pueden usarse para predecir de manera efectiva valores de VPP, sin embargo, cuando se necesita la mayor efectividad y precisión, el modelo híbrido FA-ANN sería la mejor opción. (Huang, Koopialipoor, & Armaghani, 2020)

Un último estudio por considerar en nuestros antecedentes sería el presentado por Nazanin Fouladgar, Mahdi Hasanipanah y Hassan Bakhshandeh Amnieh, y titulado ***"Application of cuckoo algorithm to estimate peak particle velocity in mine blasting"***, publicado en Londres, Inglaterra durante el año 2016 y disponible de manera virtual en: https://doi.org/10.1007/s00366-016-0463-0. Este estudio resulta importante porque relata de manera clara y concisa la necesidad de presentar alternativas de solución a la poca precisión ofrecida por los modelos y formulaciones empíricas, es por ello por lo que a continuación se presentará un resumen del estudio. En el campo de la minería y la ingeniera geotecnia, las operaciones de voladura son consideradas como el método más efectivo para la fragmentación de macizos rocosos, Sin embargo, algunos problemas ambientales son producidos al momento de la voladura, como vibraciones en el suelo y en el aire, fragmentos de roca eyectados, desplazamientos, etc., estos efectos resultan inevitables, pero pueden ser minimizados y controlados. Dentro de estos efectos no deseados, las vibraciones en el suelo son consideradas como el más importante, debido a que puede afectar de manera adversa a las estructuras cercanas al área de voladura, debilitar al macizo rocoso colindante e incluso ser molesto para las comunidades cercanas. Por lo mencionado, resulta sumamente necesario predecir de manera precisa las vibraciones en el suelo para minimizar los problemas ambientales que genera la voladura. Las vibraciones en el suelo pueden ser evaluadas en términos de Velocidad pico partícula (VPP), aceleración, frecuencia y desplazamiento; dentro de todos estos parámetros, VPP es el que mejor describe las vibraciones y por ende el comúnmente usado. Un numero grande de investigadores han

propuesto diferentes ecuaciones empíricas para estimar valores de VPP producidos durante la voladura. Sin embargo, la capacidad predictiva de etas formulaciones es generalmente pobre y limitada, es por ello, que resalta una vez más la necesidad de desarrollar modelos más acertados para la predicción de VPP. Para superar estas limitantes, técnicas de computación han sido usadas y desarrolladas por muchos investigadores. Este articulo propone un nuevo algoritmo basado en el algoritmo Cucko Search con la finalidad de crear una ecuación precisa para predecir vibraciones en el suelo producidas por operaciones de voladuras en la mina de cobre Miduk en Irán. Con fines evaluativos y comparativos, algunas ecuaciones empíricas fueron desarrolladas de igual manera en este estudio. Se utilizó información de 85 diferentes voladuras, donde se almaceno la máxima carga por retardo de cada voladura, y la distancia en metros desde el punto de monitoreo y la cara libre de la voladura; además de que se almacenó el valor de VPP de cada una de estas 85 voladuras. Para evaluar la nueva formulación propuesta y las formulaciones empíricas, se utilizó dos indicadores de desempeño, la raíz el error mínimo cuadrado (RMSE por sus siglas en inglés) y el coeficiente de correlación múltiple (R2). Comparando los valores predichos por los modelos, se demostró que la ecuación propuesta por este estudio es más confiable prediciendo valores de VPP que las ecuaciones empíricas. (Fouladgar, Hasanipanah, & Bakhshandeh Amnieh, 2017)

2.2. Bases Teóricas de la Investigación

2.2.1. Vibraciones Causadas por la Voladura

2.2.1.1. Definición

Cuando una carga explosiva es detonada en el interior de un taladro perforado, se generan y distribuyen esfuerzos dinámicos de alta intensidad alrededor del taladro, este fenómeno se explica debido a la repentina expansión de gases confinados en el interior del taladro y a la presión que ejercen sobre el macizo rocoso. Las ondas de tensión transmitidas a la roca colindante generan un movimiento ondulatorio en el suelo y la energía tensional que se transporta a través de estas ondas fragmenta el macizo rocoso debido a distintos mecanismos de rotura como pueden llegar a ser: trituración, fracturamiento radial o rotura de reflejo ante una cara libre. Cuando la intensidad de los esfuerzos tensionales que

viajan en la onda decae al nivel donde dejan de ocurrir deformaciones en el macizo rocoso, las ondas se propagan como una onda elástica, oscilando las partículas por las que viajan. El comportamiento de estas ondas simula el efecto de las ondas que se forman al dejar caer una roca sobre un estanque de agua. (Álvarez-Vigil, González-Nicieza, López Gayarre, & Álvarez-Fernández, 2012). Estas ondas en la zona elástica son conocidas comúnmente como vibraciones en el suelo. El movimiento ondulatorio se extiende concéntricamente desde el punto de la voladura hacia todas las direcciones y se va atenuando debido a la perdida de energía ocasionada por la distancia recorrida o por el encuentro con alguna otra masa de mayor dimensión que su medio. A pesar de que, las vibraciones se van atenuando exponencialmente con la distancia; el uso de grandes cantidades de explosivos, como en el caso de la minería, puede significar la generación de la energía suficiente como para amenazar la integridad y seguridad de las estructuras aledañas. El daño en estructuras se genera debido a que estas pueden verse expuestas a esfuerzos dinámicos que sobrepasan su resistencia. (Siskind, Stagg, Kopp, & Dowding, 1980)

2.2.1.2. Factores que influyen en la generación de vibraciones

A. Diseño Geométrico de la Voladura

a) Diámetro del taladro

Diferentes factores influyen en la selección del diámetro de perforación a utilizarse, como podrían ser: el grado de fragmentación deseado, el nivel de energía que se desea concentrar; o incluso, factores económicos como las dimensiones de los equipos que se posean en la unidad minera.

b) Burdem

Se denomina Burdem a la distancia entre algún taladro y la cara libre existente en la plataforma de voladura o la que se formará en medio de la voladura mientras detona cada fila. Esta distancia se mide perpendicularmente a la línea de isotiempo o de salida de material. El valor del burdem se debe elegir y calcular cuidadosamente puesto que este influirá en el grado de interacción de ondas generadas en cada taladro.

c) Espaciamiento

Se denomina espaciamiento a la distancia entre cada taladro paralela al diseño del banco de minado y perpendicular al Burdem (Delgado Ponce, 2014).

d) Altura de Banco y Largo de Taladro

La altura de banco viene generalmente determinada por los parámetros de la operación minera; desde el tamaño de bloque con el que se diseñó el modelo de bloques del yacimiento, que viene determinado por la capacidad de equipos con las que cuenta la mina. En el Perú, para minas de cobre, usualmente se trabaja con alturas de banco de 15 metros y una sobre perforación de 1.5 metros (Delgado Ponce, 2014). Es importante considerar las desviaciones que se generan al momento de perforar los taladros.

e) Taco

Se denomina Taco al material estéril que se utiliza para confinar la energía del explosivo durante la tronadura. La cantidad y dimensiones del material a utilizarse están definidos en función a los resultados esperados. La longitud del taco decidirá en gran parte el grado de confinamiento de la energía y por ende el grado de fragmentación. El material que comúnmente se utiliza es el mismo material detrítico producto de la perforación, pero es recomendable utilizar piedra chancada para alcanzar un mejor confinamiento. Se puede calcular la longitud de taco como un múltiplo del diámetro del taladro (Jimeno, Jimeno, & Francisc, 1995) o como un múltiplo del Burdem (Pfleider, 1972).

B. Secuencia de Salida de la Voladura

Un Sistema de retardos o secuencia de salida asegura que los taladros detonen en distintos instantes y no todos al mismo momento. Así mismo, los valores más altos de las vibraciones generadas son producto de solo aquel taladro que contenga la mayor carga de explosivo en todo el ordenamiento del proyecto de voladura, es por esta razón que, en las formulaciones empíricos y los modelos propuestos, se considera como parámetro de entrada la cantidad máxima de explosivo que se haya considerado utilizar en algún taladro. La carga máxima detonada afecta directamente los valores de Velocidad Pico partícula (Tosun, 2020).

C. Características de los Explosivos

Si bien es cierto, la mayor carga de explosivo utilizada durante la voladura posee gran influencia en los valores de Velocidad Pico Partícula que se obtendrán durante la detonación, otras características del explosivo mismo poseen gran influencia en la generación de vibraciones. En esencia, la presión de detonación, la cantidad de energía por kilogramo, el volumen normal de gases, la potencia relativa y la velocidad de detonación; son características de los distintos agentes de voladura o explosivos utilizados comúnmente en minería, y que pueden tener gran influencia en la generación de ondas o vibraciones en el macizo rocoso.

D. Propiedades Físicas y Mecánicas de la Roca

Conocidos por algunos autores como factores no controlables, las propiedades del macizo rocoso influyen de manera directa en la generación de vibraciones durante la voladura, puesto que detallan las características del medio en el que se desarrolla el fenómeno de las vibraciones. A continuación, se describen las principales propiedades del macizo rocoso que tienen influencia en el comportamiento de las vibraciones.

a) Módulo de Young

La deformación alcanzada por una roca está definida por las propiedades elásticas de esta, y está directamente relacionada a los esfuerzos a los que se somete por la expansión gases dentro de un taladro. El módulo de Young o de elasticidad (E) puede definirse como la relación del esfuerzo en un solo eje con la deformación en ese mismo eje; entonces, podríamos definir al módulo de Young como la medida de la cantidad de deformación que una roca puede alcanzar antes de fallar. Es posible estimar el módulo de Young de cualquier roca por medio de ensayos físicos y empíricos simples que resultarán en valores con una precisión del ± 20%, lo suficientemente preciso para realizar cualquier tipo de diseño o calculo relevante. (Farmer, 1968)

b) Resistencia de la Roca

La mayor intervención de la resistencia del macizo se puede determinar en el intento de controlar el grado de daño a estructuras cercanas y el sobre quiebre que puede generarse en el macizo rocoso por la detonación. Este

control puede lograrse en parte si se evalúan las relaciones entre densidades y velocidades, tanto del explosivo como de la roca. Las rocas con mayor resistencia necesitaran factores mayores de energía para obtener buenos resultados de fragmentación (Lopez Jimeno, 1982). Esto aplica a la mayoría de los macizos rocosos con alta resistencia, solamente se excluyen de esta regla a aquellos que contengan muchas familias de diaclasas.

c) Densidad de la Roca

Uno de los factores con mayor influencia a la hora de obtener resultados óptimos de fragmentación vendría a ser la densidad de la roca, esto debido a que, a mayor masa de roca por mover en función de un volumen, será necesario un mayor grado de energía por aplicar en el macizo a romper (Lopez Jimeno, 1982). Otro caso claro en el que la densidad juega un papel importante, es cuando el macizo posee muy pocas o ninguna familia de juntas, resultando en una mayor densidad (masa por volumen) de a roca y resultando en un mayor grado de energía necesario para fragmentar el macizo.

d) Porosidad

Podemos diferenciar dos tipos de porosidad: La intergranular, cuya distribución alrededor de toda la roca es homogénea y provoca una atenuación de la onda de choque generada y la reducción de la resistencia dinámica a la compresión. Y la porosidad post-formación, que es causada por la formación de acuíferos en el macizo rocoso; estas cavidades afectan la eficiencia de la tronadura, especialmente cuando se opta por usar explosivos bombeables y a granel. Y que pueden generar una rápida caída de la presión generada por la expansión de gases al momento de comunicar la perforación con las cavidades. (Delgado Ponce, 2014)

2.2.1.3. Velocidad Pico Partícula

Las vibraciones generadas por la voladura pueden evaluarse y expresarse de distintas maneras, ya sea velocidad pico partícula (VPP), aceleración, frecuencia o amplitud. Dentro de todos estos parámetros, la velocidad pico

partícula es la que mejor describe el fenómeno de las vibraciones en el suelo (C.H. Dowding, 1992). La velocidad pico partícula puede considerarse como un índice que permite medir las vibraciones del suelo inducidas por la voladura, así como un importante indicador en el control de daños a estructuras aledañas a los proyectos de voladura (Kahriman, 2002).

A. Fórmulas Empíricas

Un gran número de investigadores han propuesto a lo largo de los años diferentes ecuaciones empíricas para estimar valores de VPP inducidos por el proceso de voladura. Un común denominador en todas estas ecuaciones empíricas es el uso de la máxima carga en la voladura y la distancia entre el punto de medición y la voladura como parámetros independientes en el cálculo de los valores de VPP.

a) Ecuación USBM por Duvall

Duvall (1962) y sus compañeros de la United States Bureau of Mines concluyeron que la distancia lineal necesita ser dividida por la raíz cuadrada de la máxima carga. La relación correspondiente se expresa de acuerdo con lo mostrado en la **Ecuación (1).**

$$v = K\left(\frac{R}{\sqrt{Q_{max}}}\right)^{-B} \qquad (1)$$

Donde, v es la velocidad pico partícula, K y B son constantes de la mina que necesitan ser determinadas por un análisis de regresión, R es la distancia en metros del punto de medición hasta la voladura y Q_{max} es la carga máxima por retardo.

b) Ecuación del Indian Standard

La relación empírica sugerida por el Estándar de la India (1973) presenta el concepto de distancia escalar, en el que la carga es dividida por la

raíz cubica del cuadrado de la distancia lineal. Esta relación se expresa de acuerdo con lo mostrado en la **Ecuación (2)**

$$v = K\left(\frac{Q_{max}}{R^{2/3}}\right)^{B} \quad (2)$$

Donde, v es la velocidad pico partícula, K y B son constantes de la mina que necesitan ser determinadas por un análisis de regresión, R es la distancia en metros del punto de medición hasta la voladura y Q_{max} es la carga máxima por retardo.

c) Ecuación Langefors – Kihlstrom

Langefors y Kihlstrom (1978) propusieron una variante a la ecuación de la USBM, la cual definieron de acuerdo con lo mostrado en la **Ecuación (3)**.

$$v = K\left(\sqrt{\frac{Q_{max}}{R^{2/3}}}\right)^{B} \quad (3)$$

Donde, v es la velocidad pico partícula, K y B son constantes de la mina que necesitan ser determinadas por un análisis de regresión, R es la distancia en metros del punto de medición hasta la voladura y Q_{max} es la carga máxima por retardo.

d) Ecuación modificada de Ghosh-Daemen

Ghosh y Daemen (1983) propusieron que la atenuación no-elástica causa perdida de energía durante la propagación de las ondas. Este efecto inelástico conlleva a una disminución en la amplitud de la onda. Ellos reformularon la ecuación USBM de la manera presentada en la **Ecuación (4)**.

$$v = K\left(\frac{R}{Q_{max}^{1/3}}\right)^{-B} . e^{-\alpha R} \quad (4)$$

Donde, v es la velocidad pico partícula, K y B son constantes de la mina que necesitan ser determinadas por un análisis de regresión, R es la distancia en metros del punto de medición hasta la voladura, Q_{max} es la carga máxima por retardo y α es un factor de atenuación no-elástico.

e) Ecuación Rai and Singh

Rai y Singh (2004) propusieron una modificación al modelo de Ghosh and Daemen. Considerando en su nuevo modelo las atenuaciones elásticas y no-elásticas. Presentando el modelo predictivo representado en la **Ecuación (5)**.

$$v = K.R^{-B}.{Q_{max}}^{A}.e^{-\alpha R} \quad (5)$$

Donde, v es la velocidad pico partícula, K, B y A son constantes de la mina que necesitan ser determinadas por un análisis de regresión, R es la distancia en metros del punto de medición hasta la voladura, Q_{max} es la carga máxima por retardo y α es un factor de energía.

2.2.1.4. Monitoreo

Las vibraciones en el suelo se pueden medir usando un sismógrafo de tipo electromagnético o de tipo piezoeléctrico ubicado en una estación de medición, o bien utilizando un geófono de 3 dimensiones o triaxial que se instalan en el interior o la superficie del macizo rocoso a la distancia deseada. El monitoreo de vibraciones producidas durante la voladura permite evaluar el rendimiento general del diseño de la voladura. (Quiroz, 2015)

2.2.2. Machine Learning

2.2.2.1. Definición

Machine Learning o Aprendizaje Autónomo, es una rama de la Inteligencia Artificial, perteneciente a las Ciencias Computacionales; que le otorga a un sistema la capacidad de aprender a través de una base de datos histórica, en vez de hacerlo a través de una programación explicita. Machine Learning usa innumerables algoritmos que iterativamente aprenden de la base de datos otorgada

para de esta manera, mejorar su desempeño, describir de manera eficiente la información requerida, y predecir de manera óptima los resultados de salida esperados. Mientras los algoritmos se "alimentan" de más y más información para su entrenamiento, se hace posible producir modelos cada vez más precisos basados en esta información. (Hurwitz & Kirsch, 2018)

Una vez que entrenamos un algoritmo con una base de datos, obtenemos como resultado un modelo de Machine Learning; y cuando le proporcionas una base de datos de la misma fuente al modelo, este será capaz de devolver una predicción basada en su entrenamiento realizado con la primera base de datos. Es por ello por lo que resulta importante e imprescindible poseer una base de datos que refleje de manera acertada y significante la realidad que buscamos plasmar en nuestro modelo, y que sea lo suficientemente grande, consistente y diversa como para que el algoritmo pueda aprender de manera eficiente que decisiones tomar en distintos escenarios. (Lai, Trayer, Ramakrishna, & Li, 2012)

El Machine Learning se ha convertido, a lo largo de las dos últimas décadas, en una de las técnicas más importantes y utilizadas por compañías que buscan una nueva manera, eficiente y precisa, de tomar decisiones basadas en un historial de información y datos. La relevancia que el ML ha cobrado en los últimos años se explica principalmente por la basta cantidad de datos que las empresas recolectan día y día, y por la creciente necesidad de poder predecir situaciones complejas de manera que la incertidumbre sea minimizada.

A pesar de que la materia de ML parezca una zona desconocida y lejana para muchas personas, seguramente la gran mayoría de personas han interactuado con algún modelo de ML en determinado momento de su vida, o incluso en muchos de los casos, las personas interactúan con ellos de manera implícita en su día a día sin estar conscientes de ello. Por ejemplo, cuando usamos una plataforma de streaming de video, como Youtube o Vimeo, el algoritmo interno dentro de estas plataformas almacena información sobre nuestros gustos y preferencias, y en base a esta base de datos histórica que alimentamos cada vez que utilizamos estas plataformas, se nos recomendará nuevo contenido que posea características similares a lo visto anteriormente y satisfaga lo que el algoritmo entiende como

nuestros gustos y preferencias. Y esta es una de las ventajas que el ML ofrece sobre formulaciones predictivas empíricas, la facilidad y flexibilidad de poder actualizar el modelo predictivo cada vez que tengamos nueva información relevante que probablemente influya a las predicciones.

2.2.2.2. Conceptos clave

A. Distribución de una Base de Datos

Los algoritmos son entrenados usando ejemplos preprocesados pertenecientes a nuestra base de datos, esta división de nuestra base de datos se conoce como Training Set o Datos de Entrenamiento. Una vez entrenado el modelo, se mide su desempeño con una nueva sección de nuestra base de datos que no haya sido vista por el algoritmo anteriormente, esta división es conocida como Test Set o Datos de Prueba, y nos permite medir que tan bien nuestro modelo es capaz de predecir información acertada en base a nuevos datos. Existe una tercera división denominada Cross-Validation Set o Datos de Validación, la cual nos permite hacer pruebas y modificar el modelo con la finalidad de mejorar nuestra precisión sin que se "acostumbre" o ajuste de más a nuestro Test Set. Existen recomendaciones por distintos autores sobre en qué magnitudes o porcentajes dividir nuestra base de datos, sin embargo, no existe una regla escrita y que se debe seguir tajantemente, ya que esta decisión dependerá de muchos factores como el tamaño total de nuestra base de datos y nuestros recursos computacionales disponibles. Una relación comúnmente aceptada es la de 60% para el Training Set, 20% para Cross-Validation Set y 20% para Test Set. (Dobbin & Simon, 2011)

El desempeño de nuestros modelos dependerá de que tan bien puedan generalizar la realidad, y por ende es muy importante tener una Training Set que refleje las características de la realidad. Además, se demostró que el tamaño de las bases de datos es un factor decisivo para obtener óptimos desempeños en la generalización de los modelos desarrollados. (Xu & Goodacre, 2018)

B. Sobreajuste y Subajuste

Al momento de entrenar un algoritmo podemos llegar a dos tipos de resultados no deseados. El overfitting o por su traducción al español "sobreajuste" ocurre cuando el modelo es capaz de reflejar de manera casi perfecta los datos de nuestro Training Set, pero falla rotundamente cuando se le presenta nuevos datos, como podría ser el Test Set, entonces decimos que nuestro modelo se ha ajustado de más a lo aprendido y no es capaz de generalizar una predicción. Los problemas de sobreajuste ocurren usualmente cuando el modelo que se desea ajustar es tan complicado que los algoritmos buscarán ajustarse a todos los datos, resultando en soluciones complejas que no reflejan la realidad (Kishore, 2018). Otra causa del overfitting, pueden llegar a ser datos atípicos que escapan de los rangos considerados normales en una base de datos y que "obligan" al modelo a buscar soluciones complejas alejadas de la realidad.

Por su parte, underfitting o por su traducción al español "Subajuste" ocurre cuando el modelo no ha sido capaz de aprender lo suficiente y por ende nos entregará pésimos resultados, evaluándolo tanto en el Training Set como en el Test Set. Las principales razones por que un modelo puede caer en underfitting, son la poca complejidad del algoritmo seleccionado o el hecho de no haberle dado el tiempo o iteraciones suficientes de aprendizaje al algoritmo.

Idealmente se busca entrenar un modelo que encuentre un equilibrio entre estar "sobre ajustado" y estar "sub-ajustado". Esta tarea puede resultar complicada en práctica, pero no imposible (Brownlee, 2016).

2.2.2.3. Categorías o Tipos de Aprendizaje

A. Aprendizaje Supervisado

Supervised Learning o Aprendizaje Supervisado, es una técnica de aprendizaje en la que al algoritmo se le entrega la información de entrada junto con los valores de salida deseados, y la tarea principal es aprender una regla general que relacione los datos de entrada con los de salida. Una base de datos está formada por determinado número de ejemplos y a su vez, cada ejemplo está conformado por una serie de parámetros y su determinada etiqueta, que se utiliza

como valor de salida deseado. Cuando las etiquetas son valores discretos, podemos hablar de un problema de clasificación; por el contrario, si las etiquetas son valores continuos, el problema es considerado como de regresión. (Hurwitz & Kirsch, 2018)

Un claro ejemplo y aplicación de Supervised Learning sería un modelo desarrollado con la finalidad de predecir el clima. Imaginemos que poseemos una base de datos histórica de muchos años, con diferentes parámetros medidos cada día, como humedad en el aire, temperatura y velocidad del viento; y además sabemos cómo fue el clima en cada uno de estos días: soleado, lluvioso, nublado, etc. Nosotros le entregaríamos al algoritmo toda esta base histórica de parámetros con su respectiva etiqueta (tipo de clima), y el algoritmo se encargaría de encontrar una relación entre todos los parámetros entregados, y nuestra etiqueta, lo cual nos permitiría predecir cómo será el clima durante determinado día, conociendo solamente los parámetros ya mencionados. Ahora, podríamos pensar que cualquier técnico de algún instituto meteorológico podría predecir el clima con alguna formulación empírica que considere estos parámetros, pero ¿Qué pasaría si en lugar de 3 parámetros, se tuviera que considerar 500 parámetros? La importancia del aprendizaje supervisado reside en su capacidad de encontrar una correlación entre cuantos parámetros sean necesarios, basándose en información histórica real.

B. Aprendizaje No Supervisado

Unsupervised Learning o Aprendizaje No Supervisado, es una técnica de aprendizaje en la que el algoritmo es entrenado con una base de datos sin etiquetar, y cuya principal tarea es la de encontrar patrones desconocidos y estructuras en común entre todos los datos entregados, descifrando por sí solo la información. El aprendizaje no supervisado nos ayuda en problemas en los cuales tenemos grandes cantidades de datos y no podemos clasificarlos manualmente. Conocido como "Clustering" o agrupamiento, los algoritmos de Aprendizaje No Supervisado nos ayudan a descubrir grupos de datos que satisfagan condiciones en común. (Smola & Vishwanathan, 2008)

Un claro ejemplo de Aprendizaje no Supervisado, podrían ser los modelos aplicados a segmentación de mercado. Imaginemos que abriremos un negocio en

una nueva ciudad, y en el cual venderemos determinado producto. Para realizar un estudio de mercado y determinar nuestro público objetivo, decidimos adquirir una base de datos sobre la población de determinada ciudad, la cual contiene información de una muestra considerable de la población, conteniendo datos como género, edad, dirección, educación, ingresos, etc. Al nosotros no ser expertos en segmentación de mercado, decidimos desarrollar un modelo de Clustering, que, al alimentarlo con toda esta base de datos, separará a las personas en la cantidad de grupos que encuentre por conveniente o que nosotros especifiquemos, y que dentro de cada uno de estos grupos todas las personas compartan características fuertemente relacionadas.

De esta manera, el aprendizaje no supervisado lleva a cabo un proceso iterativo de análisis de información sin intervención humana.

C. Aprendizaje por Reforzamiento

Reinforcement Learning o aprendizaje por reforzamiento, es una técnica de aprendizaje en la que los algoritmos aprenden por un proceso iterativo de prueba y error, siendo “recompensados” por tomar decisiones consideradas correctas o acertadas, y siendo “penalizados” por tomas decisiones consideradas desacertadas. El aprendizaje por refuerzo difiere del aprendizaje supervisado, en el hecho que este no aprende de una base de datos etiquetada, sino aprende de la experiencia y de la retroalimentación que el usuario le entregue explícita o implícitamente como resultado de sus decisiones.

Una de las aplicaciones más comunes de aprendizaje por refuerzo son las ciencias robóticas. Tomemos el ejemplo de un robot intentando subir unas escaleras, si el robot falla en el primer intento, el algoritmo buscará una secuencia alternativa de pasos para intentarlo nuevamente, la información será recalibrada hasta que el robot tenga éxito en subir las escaleras. De esta manera, el robot solo aprende de la secuencia de pasos exitosa.

El algoritmo de aprendizaje debe ser capaz de descubrir una asociación entre la meta y la secuencia de eventos que lleven a un resultado exitoso (Hurwitz & Kirsch, 2018).

2.2.2.4. Algoritmos de Machine Learning

A. Regresión Lineal

Regresión lineal se puede definir como una metodología estadística en la que se busca modelar la relación entre dos tipos de variables, específicamente, la relación entre una o más variables independientes o predictoras y una variable dependiente o de respuesta (Kishore, 2018). Por ejemplo, podemos afirmar que la temperatura es directamente proporcional al número de helados vendidos. Si la temperatura incrementa, el número de helados vendidos también incrementará. En este ejemplo, el número de helados vendidos es la variable dependiente, y puede ser predicho basándonos en la temperatura (variable independiente).

A continuación, se presentarán algunos conceptos clave, necesarios para entender el concepto de regresión lineal.

a) Correlación

Del ejemplo previo, podemos notar que el número de helados vendidos está directamente positivamente correlacionado con la temperatura, puesto que, al incrementar la temperatura, las ventas de helado incrementan. También la correlación puede ser negativa; por ejemplo, si el precio de determinado producto incrementa, las ventas de este bajarán. (Altman & Krzywinski , 2015)

b) Causalidad

Para entender el concepto de causalidad, invirtamos el escenario planteado en nuestro ejemplo. Consideremos que la temperatura incrementa cuando las ventas de helados incrementan. Intuitivamente, podemos afirmar con toda la confianza posible que la temperatura no está controlada por las ventas de helados, mientras el escenario contrario si es cierto. Esto es un claro ejemplo de causalidad, en el que un concepto es causal o influencia a otro concepto, mas no viceversa. (Altman & Krzywinski , 2015)

c) Estructura Básica

Una regresión lineal simple es representada siguiendo la forma de la **Ecuación (6)**.

$$h_\theta(X) = b + \theta * X \qquad (6)$$

Donde $h_\theta(X)$ representa la variable dependiente que buscamos predecir, X es la variable independiente, b es el coeficiente de error o bias term, y θ es el coeficiente asignado a la variable independiente o weight.

d) Coeficiente de Error

Introduzcamos el concepto de Coeficiente de Error presentando un ejemplo. Consideremos que necesitamos predecir el valor de distintas propiedades ($h_\theta(X)$) en función a la cantidad de habitaciones que posean (X). Y observamos que el precio base de cualquier propiedad es 1000USD y que por cada habitación que posea, el precio incrementa en un factor de 0.75. Quedando el modelo representado por la **Ecuación (7)**.

$$h_\theta(X) = 1000 + 0.75 * X \quad (7)$$

El coeficiente de error será el valor que asumirá la variable dependiente cuando todos los variables independientes sean cero o no se posea información sobre la misma.

Podemos decir que el coeficiente de error es un término que representa fluctuaciones aleatorias, falta de información o medidas atípicas, permitiéndonos realizar predicciones cuando no tenemos información necesaria de las variables independientes, y generalizar el modelo de manera adecuada (Lyman & Longnecker, 2016).

e) Coeficientes asignados a las variables independientes

Son los valores calculados y asignados a cada una de nuestras variables independientes, los cuales le dan cierta "puntuación" o peso de influencia sobre la variable dependiente. En caso exista más de una variable dependiente (Regresión múltiple), la meta de la regresión lineal es encontrar los coeficientes adecuados que representen de mejor manera el nivel de correlación de todas las variables con la variable dependiente.

f) Regresión Múltiple

La regresión múltiple o multivariable, como su nombre lo sugiere, implica encontrar la correlación entre más de una variable independiente y la

variable dependiente. En la práctica, casi siempre existen múltiples variables influenciado a la variable dependiente, lo que significa que la regresión múltiple es más común que una regresión simple. (Kishore, 2018)

El mismo ejemplo mencionado con anterioridad, puede presentarse como un caso de regresión múltiple. Consideremos que las ventas de helado, ya no se ven influenciadas solo por la temperatura del día, sino ahora nuestra variable dependerá del precio del helado, estación del año que nos encontremos, día de la semana y de la temperatura del día.

Este tipo de problemas se puede representar como el modelo matemático presentado en la **Ecuación (8)**.

$$h_\theta(X) = b + \theta_1 * X_1 + \theta_2 * X_2 + \theta_3 * X_3 + \cdots + \theta_n * X_n \quad (8)$$

Donde θ representa el coeficiente o peso asignado a cada una de las variables independientes y b representa el coeficiente de error calculado para el modelo.

g) Regresión Polinómica

La regresión polinómica es un caso especial de regresión lineal en donde la relación entre las variables independientes y la variable dependiente se modela usando una ecuación de grado n.

Un modelo de grado 3 o cúbico se representa matemáticamente en la **Ecuación (9).**

$$h_\theta(X) = b + \theta_1 * {X_1}^3 + \theta_2 * {X_2}^2 + \theta_3 * X_3 \quad (9)$$

Muchas veces en la práctica, nuestros datos y la realidad no se ajusta a una línea recta o a un modelo linear, sino por el contrario presentan tendencias exponenciales, es por ello por lo que la regresión polinómica cobra especial relevancia, ya que nos permite ajustar un modelo que siga estos patrones curvos presentas en nuestros datos. (Ostertagova, 2012)

h) ¿Cómo calcular los coeficientes θ y b?

Cuando tenemos un problema de regresión lineal simple, donde solo se tiene una variable independiente y se busca modelar de manera lineal los datos, es muy sencillo estimar la pendiente de la recta y el intercepto con el eje "y" de un grupo de datos. Sin embargo, en la práctica, la mayor parte de problemas aplicativos buscan encontrar la correlación entre muchas variables independientes con la variable dependiente, y que rara vez presentan un comportamiento lineal. Es entonces que los algoritmos de optimización cobran vital importancia a la hora de solucionar este tipo de problemas. Estos algoritmos, a diferencia del método convencional estadístico, nos permiten trabajar sin problemas con grandes cantidades de variables independientes y modelar sin inconvenientes ecuaciones polinómicas. Todo esto es logrado en base al principio de minimizar una función objetivo, que en este caso sería la función de costo (Cost Function), y que esta a su vez representa el error entre los datos reales y los datos predichos. Existen un sinnúmero de algoritmos de optimización, pero uno resalta por su simpleza y efectividad; "Gradient Descent" es el algoritmo por excelencia utilizado en este tipo de problemas y el comúnmente utilizado en distintos campos aplicativos.

i) Cost Function

Es una función que mide el error en nuestro modelo en base a los valores seleccionados para los coeficientes. Nuestra meta será la de minimizar este error con la finalidad de obtener los valores más acertados para los coeficientes. Para poder medir este error podemos utilizar diferentes métricas como la media del error cuadrado (MSE por sus siglas en inglés) o la media del error absoluto (MAE por sus siglas en inglés) (Qi, Du, Sabato, Ma, & Lee, 2020). Sin embargo, la primera opción mencionada es la que se usa con mayor frecuencia, y se expresa utilizando la **Ecuación (10).**

$$J(\theta, b) = \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta\left(x^i\right) - y^i\right)^2 \qquad (10)$$

Donde y^i representa a los valores reales, y $h_\theta(x^i)$ representa a las predicciones hechas en base a determinados valores de θ y b, y n representa el número total de ejemplos existentes en nuestra base de datos.

Entonces nuestro objetivo es el de minimizar la función $J(\theta, b)$ en función de θ y b. Y para lograr esto, necesitamos del algoritmo Gradient Descent.

j) Gradient Descent

Es un algoritmo iterativo de optimización que nos permite encontrar valores para determinados parámetros (θ y b) de una función ($h_\theta(X)$), de tal manera que se minimice una función objetivo ($J(\theta, b)$).

Para darnos una idea del funcionamiento de este algoritmo, imaginemos un valle y una persona, la cual no tiene idea alguna de que dirección tomar con tal de llegar al fondo del valle. La persona comienza a bajar la pendiente, avanzando grandes distancias cuando la pendiente es elevada y dando pequeños pasos cuando la pendiente se aliviana y se aproxima a la parte más baja del valle. En cada uno de los pasos que la persona da, evalúa su posición actual y decide qué dirección tomar, y solo se detiene cuando haya llegado al fondo del valle. En esta analogía, la posición de la persona representaría el valor de nuestra Cost Function y el fondo de valle representaría el mínimo global que puede alcanzar esta función, en donde los coeficientes seleccionados serán los ideales que entreguen menor error en las predicciones. (Menon, 2018)

Ahora veamos el enfoque matemático detrás del funcionamiento de este algoritmo.

Inicialmente debemos darles algún valor a nuestros coeficientes para tener un punto de partida del cual comenzar a iterar hasta minimizar la función objetivo. Esta inicialización de valores puede ser aleatoria o simplemente darle un valor igual a cero a todos coeficientes. Adicionalmente debemos definir el Ratio de aprendizaje o Learning Rate (α), siendo este un parámetro que controlar que tanto varía el valor de los coeficientes en cada iteración. Este parámetro no puede ser ni muy grande, por que podríamos caer en un bucle infinito en el que no logramos alcanzar el mínimo de la función debido a que

los pasos son muy grandes; ni muy pequeño, porque tardaríamos demasiado en llegar al mínimo de la función (Yedida & Saha, 2019). Para entender de mejor manera este concepto, volvamos a nuestra analogía. Imaginemos que la persona tiene una catapulta que lo eyecta a no menos de 1 km de distancia, lo que resulta en la persona pasando de un lado a otro del valle sin nunca llegar al fondo; ahora imaginemos que en vez de una persona es una hormiga la que desea llegar al fondo del valle, esta podría tardar años en llegar. Por estos motivos, resulta de vital importancia encontrar un equilibrio entre precisión y uso del tiempo, y una elección correcta de α nos permite esto.

Una vez que tengamos valores iniciales para nuestros coeficientes, calculamos el error entre los valores reales y los valores predichos con nuestros valores iniciales, o el valor de la función objetivo en ese punto. Una vez calculado este valor, debemos calcular la derivada de la Cost Function en función de los coeficientes. Este valor nos indicará en qué dirección avanzar.

Ahora, debemos actualizar los valores de los coeficientes, utilizando las Ecuaciones (11) y (12).

$$\theta := \theta - \alpha\left(\frac{d}{d\theta}J(\theta, b)\right) \tag{11}$$

$$b := b - \alpha\left(\frac{d}{db}J(\theta, b)\right) \tag{12}$$

Repetimos este proceso hasta converger en el mínimo de nuestra función objetivo, resultado en valores óptimos para θ y b.

B. Árboles de Decisión

El algoritmo conocido como Decision Trees o Árboles de Decisión, es un algoritmo de aprendizaje que pertenece a la rama del aprendizaje supervisado en el que se busca aprender una serie de decisiones anidadas en base a una Base de datos de entrenamiento, y con la finalidad de poder hacer predicciones acertadas

(Tan, 2015). A diferencia de otros algoritmos, los árboles de decisión nos permiten ajustar modelos de predicción tanto como para variables categóricas, como para variables continuas.

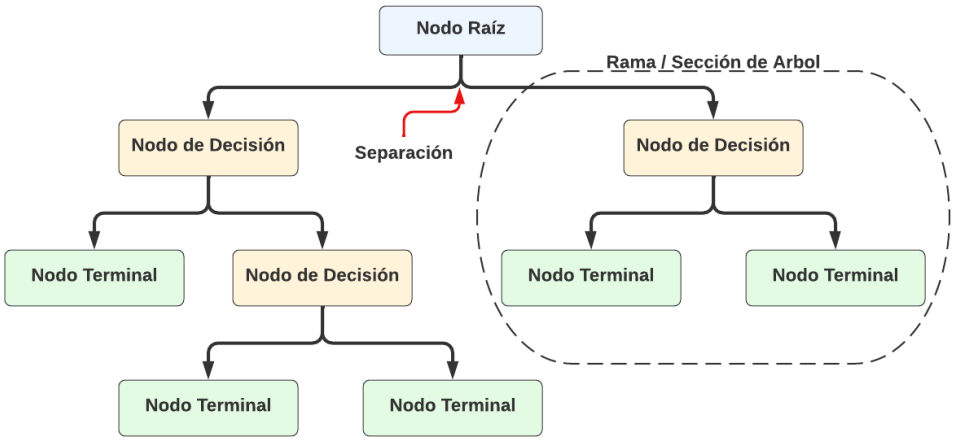
Los modelos en base a árboles de decisión se caracterizan por su flexibilidad y simpleza, resultando en ser uno de algoritmos más utilizado en la actualidad. Algunas de sus principales ventajas sobre otros algoritmos son: su facilidad de interpretación y explicación a personas no especializadas, esfuerzo reducido en el preprocesamiento de la información y que las correlaciones no lineales entre parámetros de entrada no afectan el desempeño del algoritmo. (Kotu & Deshpande, 2015)

a) Componentes de un Árbol de Decisión

Para entender el funcionamiento de un árbol de decisión, resulta importante en primera instancia definir la estructura de este, para ello presentaremos y definiremos cada uno de sus elementos, y presentaremos una representación gráfica de esta en la **Figura *1***.

Figura 1

Componentes de un Árbol de Decisión



Nota. Elaboración propia.

Nodo Raíz: también conocido como root node, representa la población o muestra entera, se divide en dos o más conjuntos homogéneos.

Separación: también conocido como splitting, representa el proceso de dividir un nodo en dos o más sub-nodos, basándose en determinada regla.

Nodo de decisión: también conocidos como decisión nodes, cuando un nodo o sub-nodos se divide en más sub-nodos, estos son denominados nodos de decisión.

Hojas/Nodos terminales: también conocidos como leafs o terminal nodes, son los nodos finales de un árbol de decisión.

Poda: también conocido como prunning, es el proceso por el cual se remueve un sub-nodo de un nodo de decisión, proceso contrario a la separación.

Rama/Sección de Árbol: también conocidos como branches, es una sección de todo el árbol de decisión.

Nodos padre e hijos: también conocidos como parent and child nodes. Cuando un nodo se divide en sub-nodos, el nodo inicial se denomina nodo padre del resto de los nodos divididos a partir de él, siendo estos sus nodos hijo.

b) ¿Cómo funciona un árbol de decisión?

Para ganar intuición en el funcionamiento detrás del algoritmo de árboles de decisión, presentemos un ejemplo aplicativo de un árbol de decisiones desarrollado. Imaginemos que deseamos predecir el sueldo un piloto de Fórmula 1, basándonos en dos distintos parámetros, la cantidad de años que posea el piloto en la máxima categoría y el número de carreras ganadas durante su trayectoria. El algoritmo aprendió que el parámetro que posee más peso a la hora de determinar el sueldo de un piloto es la cantidad de años participando en el campeonato mundial, y que el umbral divisor es de 4.5 años, entonces nuestro árbol se dividirá en dos sub-nodos, y por ende la base de datos de pilotos también se divide en dos subgrupos, aquellos que poseen 4.5 o más años de experiencia y aquellos que no. Ahora, cada uno de estos subgrupos puede dividirse entre dos o más subgrupos, pero en esta oportunidad el algoritmo aprendió que para los pilotos con menos de 4.5 años de experiencia, no es relevante la cantidad de carreras ganadas, su sueldo solo se ve influido por sus años de experiencia. Sin embargo, para los pilotos con más años de experiencia, la cantidad de carreras ganadas influirá considerablemente en su sueldo; el

algoritmo definió un umbral de 25 carreras ganadas, significando que aquellos pilotos que hayan ganado 25 o más carreras, tendrán un sueldo superior al de aquellos pilotos que posean menos de 25 carreras.

De esta manera, podemos predecir el sueldo de cualquier piloto conociendo sus años de experiencia y el número de carreras que haya ganado. Cabe resaltar, que una base de datos real puede llegar a poseer miles de parámetros, lo que resultará en arboles de decisión mucho más grandes, pero la metodología es la misma. Desarrollar un árbol de decisión se basa esencialmente en decidir qué parámetro utilizar en cada nivel el árbol y que condición utilizar para separar estos parámetros, además de saber cuándo detener el crecimiento del árbol. (Gupta, 2017)

Para seleccionar que parámetro usar en el nodo raíz, y en el resto de los nodos, debemos considerar todos los parámetros, probarlos y valorarlos en base a una función de costo o criterio de división. La división que entregue el mejor resultado (menor costo) y que separe mejor la data es seleccionada. (Gulati, Sharma, & Gupta, 2016)

c) Criterios de división

Entropía: La entropía es una medida cuantitativa del desorden o aleatoriedad en la información que se está siendo procesada. Mientras más alta sea la entropía, más difícil será obtener alguna conclusión de esa información. La entropía es usada para medir que tan informativo es determinado nodo. (Kusrini & Hartati, 2007)

La entropía se define utilizando la **Ecuación (13).**

$$E(S) = \sum_{i=1}^{c} -p_i * log_2 p_i \qquad (13)$$

Donde S es el nodo actual, c es el total de sub-nodos y p_i es la probabilidad de un evento o sub-nodo, en el nodo S. (Witten, Eide, & Hall, 2011)

En un árbol de decisión, una rama con entropía igual a cero es una hoja o nodo terminal, mientras que una rama con entropía mayor a cero necesita aún ser dividida.

Ganancia de Información: Mide que tan bien determinado parámetro, logra separar la información, en función de que tanta información entregue de la misma (Shaltout, Elhefnawi, Rafea, & Moustafa, 2014). Los árboles de decisión deben tratar de maximizar esta métrica, y los parámetros que obtengan la mayor información ganada serán divididos con prioridad.

Se puede estimar calculando la entropía del nodo antes de cierta división y substrayéndole la entropía luego de realizar determinada división, tal y como muestra la **Ecuación (14).**

$$Information\ Gain(T, X) = E(T) - E(T, X) \tag{14}$$

Gini Impurity: Es una medida de que tan seguido un elemento elegido aleatoriamente de la base de datos, será etiquetado de manera errónea (Raileanu & Stoffel, 2004). Mientras más alto sea el valor de Gini, mayor será la homogeneidad. Se calcula utilizando la **Ecuación (15).**

$$Gini = 1 - \sum_{i=1}^{c} (p_i)^2 \tag{15}$$

Donde p_i es la probabilidad de un evento y c es el total de sub-nodos en el nodo actual.

El índice Giri funciona con variables categóricas, ya que solo realiza división binaria, lo que lo limita a problemas de clasificación.

Mean Square Error (MSE): Utilizado en problemas de regresión para decidir si un nodo será dividido en dos o más sub-nodos. El algoritmo buscará minimizar este valor en cada división. MSE mide el promedio de los errores al

cuadrado, es decir, la diferencia entre el estimador y lo que se estima. En la **Ecuación (16)** encontramos la representación matemática del MSE.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(S_i - O_i)^2 \quad (16)$$

Donde, O_i son los valores reales, S_i son las predicciones o valores calculados y n es el número total de datos.

d) ¿Cuándo parar de dividir un árbol de decisión?

Los problemas reales suelen tener un gran número de parámetros que analizar, lo que resulta en un inmenso número de divisiones, lo que lleva a su vez a un árbol de dimensiones enormes. Estos árboles de gran dimensión pueden seguir creciendo indiscriminadamente hasta representar de manera casi perfecta la base de datos de entrenamiento, lo que resultaría en un problema de overfitting o sobreajuste, donde nuestro modelo no será capaz de representar la realidad cuando se le presenten datos nuevos. (Bramer, 2016)

Existen distintas formas de limitar el crecimiento de un árbol de decisión, entre ellas las más utilizadas son: el Pre-prunning o poda previa, que consiste en limitar el crecimiento del árbol, ya sea configurando en número máximo de divisiones por nodo o la profundidad máxima del árbol; otra técnica utilizada es el Post-prunning o poda posterior, consiste en remover aquellas ramas que utilicen parámetros con poca relevancia sobre la predicción o que no ayuden a minimizar la función de costo.

C. Bosques Aleatorios

Es un algoritmo que funciona por aprendizaje de ensamble por bagging, es decir, que entrena un número finito de árboles de decisión en paralelo, donde, como menciona Brownlee (2016), cada uno de estos árboles se entrena en una muestra aleatoria de la base de datos original, y donde cada árbol de decisión entregará un resultado individual, y que considerados como un conjunto nos entregan un resultado más acertado. Según Louppe (2014), la eficacia de los árboles de decisión sumada a la aleatoriedad que entrega el bagging, reducen el

grado de correlación y logran reducir el error de estimación comparado con un modelo entrenado con un solo árbol de decisión. Y como menciona Yiu (2019), un gran número de modelos relativamente poco correlacionados (árboles de decisión en este caso) operando como un todo, tendrá mejores resultados que cualquier modelo individual consistente.

a) Bagging

El término bagging deriva de la abreviatura de "Bootstrap Aggregating". Donde Bootstrap hace referencia a cualquier método de extracción de muestras aleatorio con reemplazo. Significando así, que en cada subsección de la base de datos es probable que existan datos repetidos pero seleccionados aleatoriamente. Y donde Aggregating, hace referencia a tomar la media (para modelos de regresión) o la moda (para modelos de clasificación) de todos los resultados de los árboles de decisión que han sido construidos con la base de datos, y entregarlo como predicción final. Este método de aprendizaje ayuda a reducir significativamente la varianza del modelo final, y por ende evitar el sobreajuste de este. (Biau, 2012)

b) ¿Cómo funciona el algoritmo de Bosques Aleatorios?

Para ganar intuición en el funcionamiento de este algoritmo, presentemos un ejemplo didáctico. Imaginemos que Pedro desea tomarse unas vacaciones y viajar a algún lugar durante este tiempo, sin embargo, Pedro no ha viajado nunca y, por ende, no tiene noción alguna de que tipo de lugar debería visitar durante sus vacaciones. Pedro decide pedir ayuda a varios de sus amigos para decidir donde pasará sus próximas vacaciones. Todo esto bajo la consigna de que Pedro les responderá un número determinado de preguntas y en base a sus respuestas, sus amistades recomendarán un lugar a Pedro, y el lugar recomendado con mayor frecuencia, será el elegido. El primer amigo de Pedro, le pregunta a si prefiere el calor o el frio, a lo que Pedro responde que siempre ha preferido el calor; luego Pedro es preguntado si su presupuesto supera los 1000$, a lo que Pedro responde que no. El primer amigo recomienda visitar Máncora. Un segundo amigo, pregunta nuevamente si Pedro prefiere el frio o el calor, a lo que Pedro responde nuevamente que prefiere el calor, luego le

pregunta si prefiere la playa o el campo, a lo que Pedro responde que prefiere la playa. El segundo amigo recomienda a Pedro visitar Máncora. Un tercer amigo, no cree relevante preguntar las preferencias de clima, en cambio, pregunta si Pedro quiere actividades culturales o recreativas, y si Pedro está soltero o casado; Pedro responde, actividades culturales y casado, a ambas preguntas respectivamente. El tercer amigo recomienda visitar Cuzco.

Luego de haber sido interrogado por un número finito de amigos, Pedro determina que Máncora fue la opción más recomendada y por ende la opción que reduce la variabilidad de ajustarse a sus gustos y en consecuencia la opción elegida.

Al igual que en este ejemplo, el algoritmo de Bosques Aleatorios creará un número finito de árboles de decisión considerando para cada uno de ellos un conjunto de parámetros aleatorias, a diferencia de un modelo puro de árbol de decisión, donde los parámetros se seleccionan priorizando los que separen de mejor manera el modelo y entreguen una mejor métrica de división, esto reduce el grado de correlación entre los árboles (Orellana, 2018). Es decir, un árbol de decisión puede considerar clima y presupuesto, mientras que otro considerará clima y tipo de destino. Finalmente, algo que debemos agregar a este ejemplo es que cada uno de los árboles de decisión será entrenado con un subconjunto aleatorio de la base de datos, garantizando que cada árbol individual será único, lo que reduce la varianza total del modelo. (Misra, Li, & He, 2020)

D. Potenciación de Gradiente

Potenciación de Gradiente o Gradient Boosting es un algoritmo de Machine Learning que entrega un modelo predictivo que ensamblan de manera consecutiva un conjunto de modelos predictivos débiles, usualmente árboles de decisión, para formar un modelo predictivo fuerte o lo suficientemente robusto como para entregar predicciones acertadas (Fafalios, Charonyktakis, & Tsamardinos, 2020). Utilizado tanto en problemas de clasificación como en problemas de regresión, el algoritmo Gradient Boosting crea cada árbol de manera que se corrijan los errores del árbol anterior. Una vez entrenado el modelo final, las predicciones se representarán, al igual que en al algoritmo Bosques Aleatorios,

como el valor del promedio (para problemas de regresión) o como el valor de la moda (para problemas de clasificación). El proceso de aprendizaje en este algoritmo se logra a través de la optimización de una función de costo o de pérdida (Loss Function).

El algoritmo Gradient Boosting, involucre tres elementos principales: Una función de perdida (Loss Function) para ser optimizada, un modelo débil (Weak learner) para hacer predicciones y un modelo aditivo para añadir los weak learners a fin de minimizar la función de perdida.

a) Función de Pérdida

La función de pérdida o costo, como ya vimos con anterioridad, nos permite medir el rendimiento de la predicción de determinado modelo, y por ende funciona como función de optimización para entrenar un modelo predictivo. En este algoritmo, la función de costo dependerá del tipo de problema que estemos afrontando, por ejemplo, para un problema de clasificación podemos utilizar la función Sigmoid; mientras que, para un problema de regresión, el error medio cuadrado será ideal como función objetivo a minimizar.

La función Sigmoid, es una función que nos permite convertir valores discretos a valores continuos, muy útil en problemas de clasificación, ya que podemos convertir la recurrencia de un valor discreto en las predicciones a un valor continuo de clasificación. Esta función se describe de acuerdo a la **Ecuación (17).**

$$f(z) = \frac{1}{1 + e^{-z}} \tag{17}$$

b) Modelo Débil

Los árboles de decisión simples son utilizados como modelos débiles o weak learners en el algoritmo Gradient Boosting. Estos árboles de decisión son desarrollados o construidos en base a criterios de división como el grado de impureza Gini o el RMSE. Es recomendable mantener estos árboles de decisión

lo más simple posibles, esto se puede lograr limitando el número máximo de divisiones, de nodos, o limitando la profundidad del árbol.

c) Modelo Aditivo

Los árboles son añadidos al modelo uno a la vez, y una vez que han sido añadidos no pueden ser modificados. Cada vez que se quiere añadir un nuevo árbol, se mide la función de perdida y se añade un árbol que reduzca esta función objetivo. Esto lo logramos seleccionando los parámetros correctos que logran reducir el error en nuestro modelo aditivo. Se añadirán la cantidad de árboles necesaria como para alcanzar un error aceptable o cuando se haya alcanzado el mínimo de la función objetivo, resultando en un modelo óptimo. Usualmente se conoce a esta metodología como Functional Gradient Descent o Gradient Descent con funciones. (Mason, Baxter, Bartlett, & Frean, 2000)

E. Máquina de Vectores de Soporte

Support Vector Machine o SVM, es uno de los algoritmos más populares en la rama del aprendizaje supervisado, utilizado tanto en problemas de regresión (Support vector regression) como en problemas de clasificación (Support vector classification). Sin embargo, es utilizado con mayor frecuencia en problemas de clasificación, debido a su relativa simpleza y flexibilidad. (Mechelli & Vieira, 2019)

El objetivo del algoritmo SVM es el de crear la mejor línea o borde de decisión (decision boundary) que pueda segregar o separar un espacio de n dimensiones en determinado número de clases, con la finalidad de que en un futuro podamos fácilmente determinar a qué categoría pertenece un nuevo punto. Este borde de decisión se denomina hiperplano. Para separar la data y determinar las clases en nuestra data, hay infinitas posibilidades de hiperplano que podrían ser seleccionadas. Sin embargo, el objetivo del algoritmo es encontrar un plano que posea un margen máximo entre el hiperplano y los puntos o vectores extremos, datos más cercanos entre sí, que pertenecen a distintas categorías (Jayadeva, Khemchandani, & Chandra, 2017). Estos puntos extremos se denominan Vectores de Soporte, y de aquí es que radica la nomenclatura del algoritmo. En la **Figura *2***, podemos observar cada una de las partes de un modelo de Vectores de Soporte

clasificatorio simple. Siendo el hiperplano con margen máximo el mejor borde de decisión que maximiza la separación entre los vectores o puntos más cercanos de los dos grupos de datos.

a) Vectores de Soporte

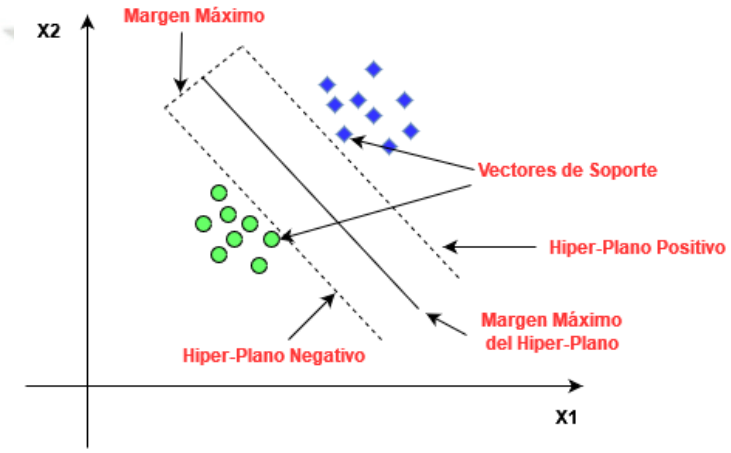
Son los datos que están más cerca al hiperplano e influencian la posición y orientación de este. Usando estos vectores de apoyo, maximizamos el margen del clasificador y si eliminásemos estos vectores, la dirección y posición del hiperplano cambiaría. Estos datos ayudan y son la base de la construcción de un modelo con SVM.

b) Hiperplano

Es el mejor borde de decisión que clasifica la información y maximiza el margen entre los vectores de apoyo. Las dimensiones de este hiperplano dependerán de la cantidad de parámetros que tengamos en nuestra base de datos, lo que significa que, si tenemos dos parámetros, el hiperplano será una línea; y si tuviésemos tres parámetros, el hiperplano será un plano de dos dimensiones. A partir de este punto es difícil imaginar las dimensiones del hiperplano, pero esta regla es constante para cualquier número de parámetros que tengamos.

Figura 2

Componentes de un modelo Máquina de Vectores de Soporte



Nota. Elaboración Propia

Sin embargo, es posible trabajar con hiperplanos de dimensión igual o incluso mayor al número de parámetros, esto para resolver problemas de mayor complejidad en los que no sea posible separar la data con modelos lineares. Es por eso por lo que existen dos grandes tipos de funcionamiento de este algoritmo.

c) Tipos de SVM

SVM lineal: se utiliza en problemas de regresión y clasificación simple, donde la data es fácilmente separable y no es necesario agregar más dimensiones a la misma.

SVM no lineal o kernel SVM: cuando la estructura de nuestra data es tan compleja que necesitamos añadir dimensiones a nuestro hiperplano para lograr separar adecuadamente la información, usamos una función denominada kernel (Satapathy, Dehuri, Jagadev, & Mishra, 2016).

d) Kernel

Un kernel (k) es una función que devuelve el resultado del producto punto entre dos vectores, añadiendo una nueva dimensión al espacio dimensional original en el que se encuentran los vectores. Existen distintos tipos de kernels. A continuación, se detallan los principales y mayormente utilizados en la práctica:

Lineal: Recomendado si la separación lineal de la data es relativamente simple. Usar un kernel lineal, equivale a no usar un kernel en lo absoluto. En la **Ecuación (18)** encontramos la representación algebraica del kernel lineal.

$$K(x, x') = x.x' \tag{18}$$

Polinómico: Es una representación más generalizada del kernel lineal. En la **Ecuación (19)** encontramos la representación algebraica del kernel polinómico.

$$K(x, x') = (x.x' + c)^d \tag{19}$$

Donde d representa el grado polinomico que deseamos alcanzar, a medida que aumenta el valor de este parámetro, los bordes de decisión se hacen menos lineales y más polinómicos.

Gaussian Radial: uno de los kernels más populares y preferidos. Usualmente usado para datos no-lineales. En la **Ecuación (20)** encontramos la representación algebraica del kernel radial gaussiano.

$$K(x, x') = \exp(-\gamma \|x - x'\|^2) \quad (20)$$

El valor de γ regula la flexibilidad del modelo, si el valor de γ es muy grande se asemejará a un kernel lineal, y mientras aumenta su valor, se obtienen modelos cada vez más complejos. Se debe tener cuidado de no sobreajustar el modelo, puesto que valores muy grandes de γ vuelven al modelo muy sensible.

Sigmoid: sirve como una función de activación. En la **Ecuación (21)** encontramos la representación algebraica del kernel sigmoidal.

$$K(x, x') = (\gamma(x.x') + r) \quad (21)$$

Posee dos parámetros ajustables, γ que ajustará la flexibilidad del modelo y r que es una constante de intercepto.

e) Regresión con Vectores de Soporte

SVM también puede ser usado como un método de regresión, manteniendo todos los principales componentes que caracterizan al algoritmo (maximizar el margen). El método Regresión con Vectores de Soporte, utiliza el mismo principio que SVM para clasificación, con solo unas diferencias menores. Primeramente, debido a que el valor de salida es un valor continuo (número real) y a que existen infinitas posibilidades de modelamiento, se vuelve extremadamente complicado obtener predicciones. Un margen de tolerancia (épsilon) es configurado antes del entrenamiento y donde deben estar los vectores de apoyo. Entonces el principal objetivo es encontrar una función que relacione una variable de entrada X y una variable de salida Y, todo esto

buscando minimizar el error que entrega un hiperplano que contenga a los vectores de apoyo en un margen determinado (épsilon). (Christmann & Steinwart, 2008)

F. Redes Neuronales Artificiales

Las redes neuronales artificiales son un algoritmo diseñado para emular el funcionamiento del cerebro humano, de modo que las computadoras aprendan a lidiar con problemas de alta complejidad al igual que lo hace el cerebro humano mediante la interrelación de millones de neuronas, y por un proceso de aprendizaje por prueba y error. Las redes neuronales constan de un conjunto de unidades de procesamiento, denominadas neuronas artificiales, conectadas entre sí para transmitirse información. La información entregada atravesará la red neuronal, se someterá a distintas operaciones y devolverá un valor de salida. Cada neurona se encuentra interconectada con otras mediante un enlace, donde el valor de la última neurona es multiplicado por un vector o matriz de pesos que incrementará o reducirá el impacto de la última neurona en la neurona subsecuente, mediante un proceso denominado propagación hacia adelante (Nunes da Silva, Hernane, Andrade, Bartocci, & Dos Reis, 2017). Uno de los beneficios que nos entrega el uso de redes neuronales es el hecho que se puede aumentar la complejidad del modelo, utilizando una función de activación o aumentando la cantidad de unidades por capa o incluso la cantidad de capas ocultas a utilizar.

Este sistema neuronal artificial aprende de forma autónoma, buscando minimizar una función objetivo que mide el desempeño de la red neuronal cuando trabaja con determinados pesos. Estos pesos se irán modificando hasta alcanzar sus valores óptimos que minimicen el error final de nuestra red. Este proceso de aprendizaje se denomina propagación hacia atrás y es muy similar a la optimización por gradient descent. (Mirjalili, 2019)

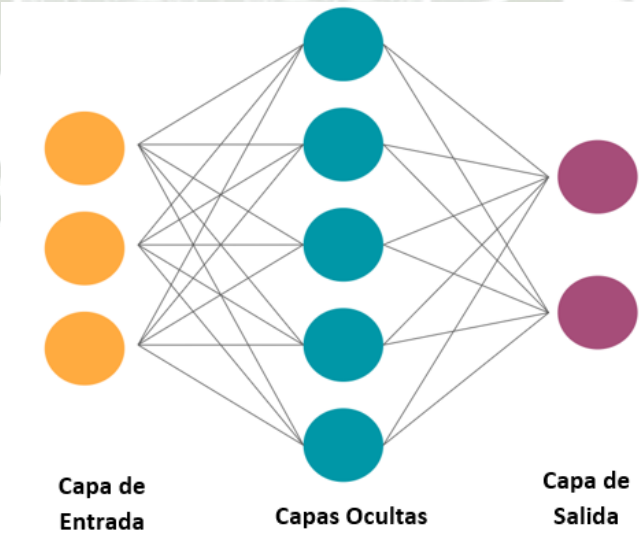
a) Estructura de una Red Neuronal Artificial

Una red neuronal está compuesta en esencia de tres tipos de layers o capas, a su vez cada capa está compuesta de unidades o neuronas, y entre cada capa, existen conexiones o pesos que interrelacionan las neuronas de una capa con los de otra. En la **Figura *3***, el conjunto de neuronas naranjas representa

nuestra capa de entrada o input layer, que sirve como entrada para la información, cada una de estas neuronas ocupará el lugar de un parámetro de nuestra base de datos (variables independientes), por ejemplo, si queremos calcular el precio de una casa, en la capa de entrada ingresaremos el número de habitaciones, la cantidad de metros cuadrados de la propiedad, cantidad de pisos construidos, etc. El siguiente conjunto de neuronas, representadas en color azul, forman nuestra capa oculta o hidden layer, aquí es donde la red neuronal buscará encontrar patrones comunes en la información antes de hacer una predicción, se denomina oculta puesto que los valores calculados en esta capa no son entregados como valor de salida al usuario. Finalmente, el tercer grupo de neuronas, representadas de color guinda, componen la capa de salida u output layer, que es donde se entrega una predicción final (variable dependiente); para problemas de regresión solo tendremos una neurona en esta capa, pero en problemas de clasificación puede existir tantas neuronas como cantidad de clases existan en el problema. (Brownlee, 2016)

Figura 3

Estructura de una Red Neuronal Artificial



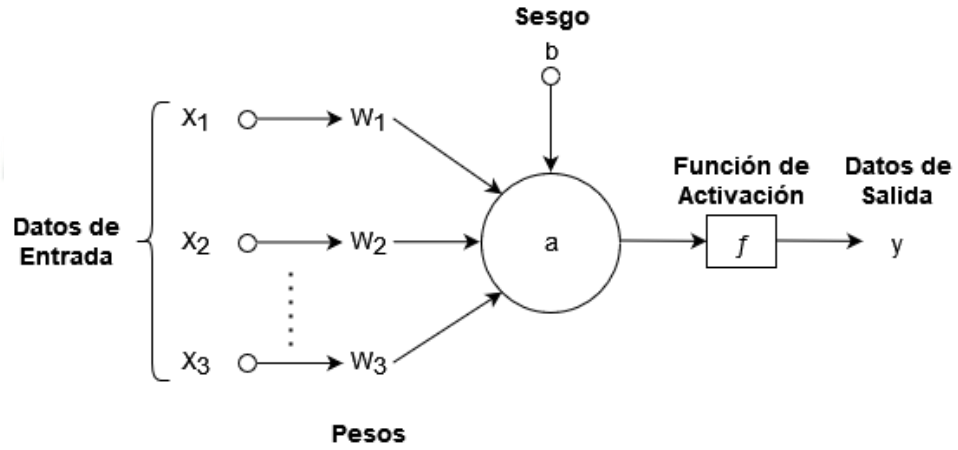
Nota. Elaboración propia

b) Forward Propagation

El proceso de propagación hacia adelante es el encargado de computar una predicción, considerando los parámetros de entrada y determinados pesos para cada unidad. Veamos a detalle cómo funciona considerando una estructura sencilla presentada en la **Figura *4***.

Figura 4

Propagación hacia adelante



Nota. Elaboración propia

Consideremos este caso en que tenemos solamente una capa de entrada y una capa de salida con una sola neurona. En la capa de entrada tenemos todos los parámetros o variables independientes y tenemos que llegar a nuestra capa de salida que representa nuestra predicción o variable dependiente. Esto lo logramos mediante los siguientes pasos:

1. Sabiendo que X_i representa cada uno de los valores de entrada, w_i representa los pesos asignados a cada uno de los parámetros y b representa un coeficiente de error o bias term. Podemos calcular el valor de a utilizando la **Ecuación (22):**

$$a = \left(\sum_{i=1}^{n} w_i * X_i\right) + b \qquad (22)$$

2. Una vez calculado el valor de a, procedemos a aplicar una función de activación, tal cual se muestra en la **Ecuación (23).** Esta función de activación puede ser una sigmoidal o una tangencial.

$$y = f(a) \tag{23}$$

De esta manera es como se relacionan distintas neuronas para calcular el valor de siguiente neurona, ahora imaginemos que tenemos capas intermedias ocultas entre la entrada y la salida, el proceso es el mismo pero la relación y complejidad del modelo aumenta. De hecho, tener más de una capa oculta ayuda a tener un alto grado de no-linealidad y resolver problemas de alta complejidad, puesto que, sin esto, una red neuronal no sería más que una función lineal gigante. (Brownlee, 2016)

En una red neuronal, cambiar los pesos de cualquier conexión tiene un considerable efecto en el resto de las neuronas y sus activaciones en las capas subsecuentes. Según Yiu (2019) esto se debe a que cada neurona en la red neuronal funciona como un pequeño modelo, y la red neuronal en su totalidad funciona como un acumulado de estos pequeños modelos cuyos valores de salida alimentan a otros modelos consecutivamente.

c) Función de Activación

Ya mencionamos la existencia de una función de activación, y su uso se debe en esencia a la necesidad de transformar señales de entrada de baja complejidad a señales de salida con mayor complejidad, con la finalidad de alcanzar modelos de alta complejidad no lineales. Otro uso importante de las funciones de activación es la de limitar las salidas en diferentes capas, o incluso antes de presentar la predicción, esto cobra vital importancia en problemas de clasificación cuando se espera entregar un valor discreto como resultado final.

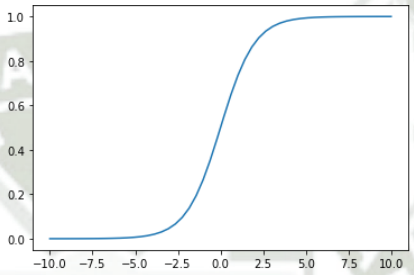
Algunas de las principales funciones de activación se presentan a continuación.

Función Sigmoid: recibe cualquier valor perteneciente a los números reales y devuelve un valor entre 0 y 1. Se representa matemáticamente de acorde a la **Ecuación (24)** y gráficamente de acorde a la **Figura *5***.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{24}$$

Figura 5

Representación gráfica de la función de activación Sigmoid.



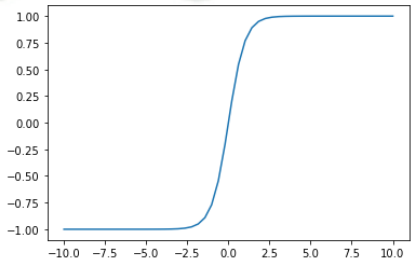
Nota. Elaboración propia

Función Tanh: toma un valor de los números reales y devuelve un valor entre -1 y 1. Se representa matemáticamente de acorde a la **Ecuación (25)** y gráficamente de acorde a la **Figura *6***.

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{25}$$

Figura 6

Representación gráfica de la función de activación Tanh.



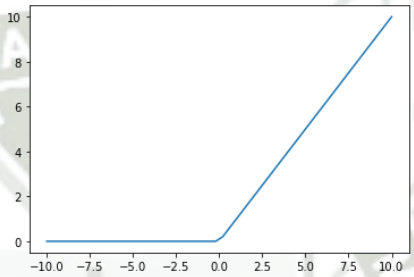
Nota. Elaboración propia

Función ReLu: Rectified Linear Unit, toma un valor de los números reales y devuelve 0 si es valor es negativo, de lo contrario devuelve el valor de entrada. Se representa matemáticamente de acorde a la **Ecuación (26)** y gráficamente de acorde a la **Figura *7***.

$$R(z) = \max(0, z) \tag{26}$$

Figura 7

Representación gráfica de la función de activación ReLu



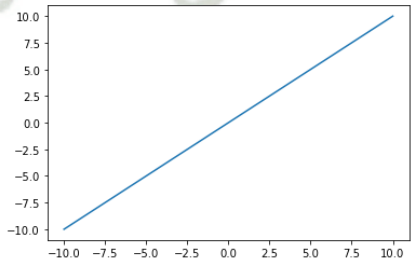
Nota. Elaboración propia

Función Lineal: también conocida como función identidad, devuelve el mismo valor entregado como entrada. Se representa matemáticamente de acorde a la **Ecuación (27)** y gráficamente de acorde a la **Figura 8**.

$$L(z) = z \tag{27}$$

Figura 8

Representación gráfica de la función de activación Lineal



Nota. Elaboración propia

d) Inicialización Aleatoria

Al igual que en el algoritmo de regresión lineal, debemos inicializar los valores de peso para computar un error inicial y en base al cual podamos comenzar el proceso de aprendizaje iterativo hasta llegar al error mínimo. Sin embargo, en una red neuronal no se puede iniciar todos los valores como ceros, porque esto causaría que todas las neuronas intermedias en las capas intermedias ocultas tengan los mismos valores, puesto que han sido calculadas con pesos iguales. Entonces, resulta imprescindible inicializar estos valores de manera aleatoria, de esta manera logramos romper la simetría entre las unidades (Goodfellow, Bengio, & Courvile, 2016). Existen distintos criterios de inicialización, pero los comúnmente utilizados incluyen distribuciones normales y escaladas para facilitar el aprendizaje.

e) Propagación hacia atrás

Una vez entendido el proceso mediante el cual los datos de entrada logran convertirse en datos de salida y de cómo debemos asignar valores iniciales al modelo. Podemos comenzar a detallar el proceso de aprendizaje mediante el cual una red neuronal logra aprender a representar modelos complejos. Este proceso de aprendizaje se logra a través de un algoritmo llamado Propagación hacia atrás o Back-propagation. En la propagación hacia adelante, procedimos desde los valores de entrada hasta los valores de salida. En la propagación hacia atrás, procedemos de manera inversa: esencialmente, cambiar los pesos en pequeños valores, empezando desde la última capa, hasta que se alcance el menor error posible. A continuación, detallamos el funcionamiento de este algoritmo.

Una vez inicializados los pesos y calculado el valor final de nuestro modelo a través de una propagación hacia adelante, podemos evaluar el desempeño de nuestro modelo en ese momento utilizando una función de costo, que determinará el error final de nuestro modelo. Ahora, resulta imposible calcular directamente el error para las neuronas intermedias, debido a que no existen valores con los cuales podamos comparar los calculados en estas neuronas. Entonces debemos propagar el error final calculado para todo el

modelo de regreso al resto de neuronas. Sin embargo, no todas las neuronas reciben igual fracción del error final, sino que reciben un error proporcional al impacto que hayan tenido en el error total, para determinar esta proporción debemos calcular la derivada parcial del error total en función de los pesos para determinada neurona. Este proceso nos indicará la pendiente en la cual debemos modificar nuestros pesos, adicionalmente, podemos multiplicarlo por un factor de aprendizaje que determinará qué tan rápido modificamos los valores y que tan preciso puede ser el aprendizaje. (Mazzur, 2015)

2.2.2.5. Indicadores de Desempeño

A. Raíz del Error Cuadrático Medio (RMSE)

Es una medida entre valores que son conocidos y valores que han sido interpolados o modelados. El uso de RMSE es muy común y es considerado un excelente indicador de desempeño para predicciones numéricas. Resalta su uso cuando se busca comparar errores de predicción entre distintos modelos, y se recomienda su uso cuando la data tiene una distribución cercana a la normal, puesto que es muy sensible a valores atípicos. (Chai & Draxler, 2014)

Se calcula, elevando al cuadrado la diferencia entre los valores reales y los calculados, sumando estas diferencias, dividiendo esta suma entre el número de datos y finalmente extrayendo la raíz cuadrada de este resultado. Para mejor entendimiento, revisar la representación matemática presentada en la **Ecuación (28).**

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(S_i - O_i)^2} \qquad (28)$$

Donde, O_i son los valores reales, S_i son las predicciones o valores calculados y n es el número total de datos.

B. Coeficiente de Determinación (R2)

Es una medida estadística que expresa la fuerza de correlación en nuestro modelo, y que tan bien nuestro modelo lograr reflejar la realidad de los datos. Esto significa que mientras más fuerte sea la correlación, mejor se podrán predecir valores de Y con respecto a X (Lyman & Longnecker, 2016). También conocido como R cuadrado, el coeficiente de determinación se calcula dividiendo la varianza explicada por el modelo evaluado, sobre la varianza total en la base de datos. Para mejor entendimiento, revisar la representación matemática presentada en la Ecuaciones (29), (30) y (31).

$$R2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{29}$$

$$SS_{res} = \sum_i (\hat{y}_i - \bar{y})^2 \tag{30}$$

$$SS_{tot} = \sum_i (y_i - \bar{y})^2 \tag{31}$$

Donde, SS_{res} es la varianza explicada por el modelo, SS_{tot} es la varianza real existente en la data, $\hat{y}_i$ representa los valores predichos, $\bar{y}$ representa la media de y, y_i representa los valores de y.

Cuanto más cercano el resultado sea a 1 o 100%, se puede afirmar que el modelo refleja de mejor manera la varianza existente en la base de datos (Rieuf, 2017).

2.2.2.6. Métricas de Importancia de Variables

A. Explicación Aditiva de Shapley (SHAP)

Es una métrica que permite determinar la contribución de las variables consideradas en un modelo predictivo, estimando el promedio de contribución en todas las posibles permutaciones de las variables. Es una métrica muy utilizada para entender el funcionamiento de los modelos de Machine Learning. (Dr. Dataman, 2019)

2.3. HIPOTESIS

Los modelos predictivos desarrollados con algoritmos de Machine Learning presentarán mejores métricas de desempeño comparados con los modelos predictivos desarrollados mediante formulaciones empíricas.

2.4. VARIABLES

Tabla 1

Operacionalización de variables

VARIABLE	INDICADOR	UNIDAD DE MEDIDA	INSTRUMENTO	FUENTE
MODELOS PREDICTIVOS EN BASE A FÓRMULAS EMPÍRICAS	Precisión (Raíz del Error Cuadrático Medio)	Unidades. En este caso al medir la precisión de las predicciones de VPP, la unidad sería: m/s	Formulación Estadística.	Naturaleza del modelo
	Coeficiente de Determinación (R2)	Porcentaje (%)	Formulación Estadística.	Naturaleza del modelo
MODELOS PREDICTIVOS EN BASE A ALGORITMOS DE MACHINE LEARNING	Precisión (Raíz del Error Cuadrático Medio)	Unidades. En este caso al medir la precisión de las predicciones de VPP, la unidad sería: m/s	Formulación Estadística.	Naturaleza del modelo
	Coeficiente de Determinación (R2)	Porcentaje (%)	Formulación Estadística.	Naturaleza del modelo

Nota: Elaboración propia.

CAPÍTULO III

3. MARCO METODOLÓGICO

La metodología por utilizarse en esta investigación consta en esencia de dos grandes partes, siendo la primera una dedicada revisión bibliográfica que nos permitirá establecer las bases teóricas de nuestro experimento, y la segunda parte que considera la experimentación con todas sus etapas. Para mayor detalle y mejor entendimiento, revisar la **Figura *9***.

Como parte inicial de la metodología, se realizará una exhaustiva revisión bibliográfica en bases de datos relevantes como revistas indexadas, repositorios de tesis nacionales e internacionales o incluso memorias de congresos; esto nos permitirá establecer los antecedentes de nuestra investigación, además de conocer los enfoques utilizados por otros investigadores a la hora de afrontar una problemática similar a la mencionada en el presente estudio, así como sentar la línea base de nuestra investigación.

Una vez terminada la primera etapa, se procederá a realizar la experimentación, la cual será detallada a continuación de principio a fin. Primeramente, y una vez tengamos una base de datos lo suficientemente considerable como para poder entrenar modelos computacionales, debemos proceder a "limpiar" la información, separando y descartando datos que escapen de la realidad, ya sea por errores en la medición o por que hayan sido alterados por factores no controlables durante el momento de la medición, de esta manera garantizamos que los modelos computacionales reflejen de mejor manera la realidad estudiada. Posteriormente, la base de datos será dividida en dos secciones, en donde el 80% de los datos serán destinados para el aprendizaje de los modelos predictivos de Machine Learning, y el 20% restante será utilizado con propósitos de validación o prueba de los modelos, es decir, se utilizará para medir que tan precisos son los modelos cuando se les presenta información "nueva".

Una vez completado este primer paso, se procederá a seleccionar los algoritmos computacionales que serán utilizados para entrenar los modelos predictivos, esta selección encuentra su base y fundamento en la revisión bibliográfica realizada con anterioridad, puesto que se tomará en cuenta aquellos modelos que hayan tenido mejor desempeño en otros casos de estudio alrededor del mundo, siendo estos: Red Neuronal Artificial y Regresión con Vectores de Soporte; además se incluirán algunos modelos que a criterio del autor podrían obtener métricas de desempeño iguales o incluso mejores a los propuestos en investigaciones anteriores, siendo estos: Árboles de Decisión, Bosques Aleatorios, Gradient Boosting Machine; finalmente se decidió incluir un modelo también considerado como parte de las

técnicas computacionales pero que encuentra sus bases principalmente en la estadística, siendo este: Regresión Múltiple.

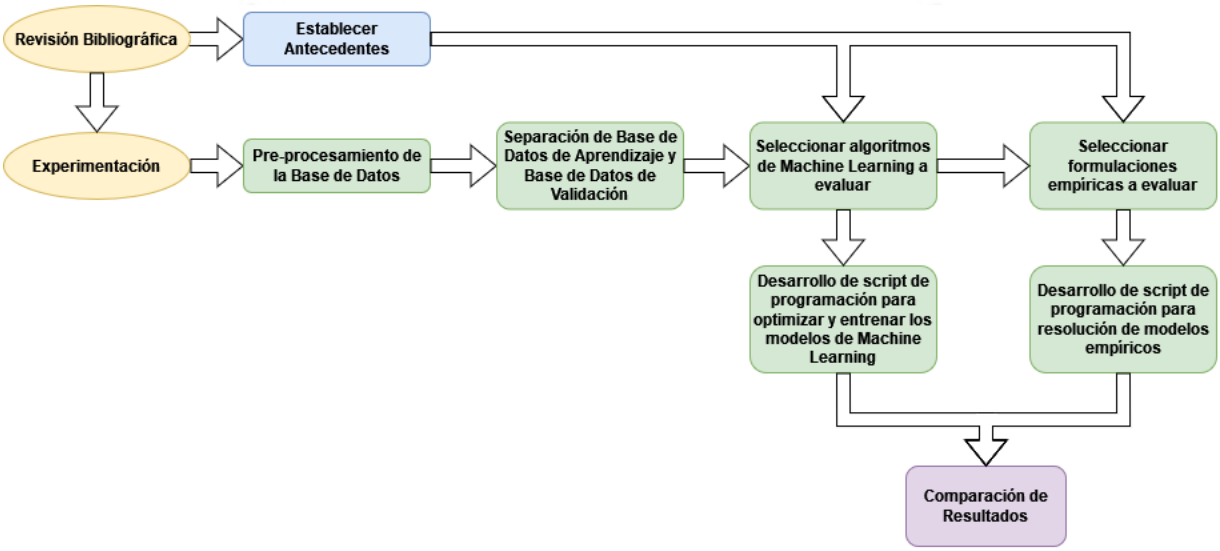
Posterior a la selección de los algoritmos a utilizar para entrenar los distintos modelos predictivos, se procederá a programar cada uno de los modelos mencionados utilizando un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) que soporte lenguaje de programación Python, en esta ocasión utilizaremos el entorno de desarrollo Spyder. Para concretar esta tarea, nos apoyaremos de distintas librerías o paquetes especializados de libre acceso, los cuales nos permiten entrenar modelos computacionales con distintos algoritmos, requiriendo de una base de datos de entrada y de una configuración de parámetros característicos de cada algoritmo; las librerías que utilizaremos serán: Scikit Learn, XGBoost, Keras y TensorFlow; además nos apoyaremos de librerías especializadas para la lectura y procesamiento de datos como serían: Pandas, Numpy. Y una librería especializada en graficar resultados, Matplotlib. Es de suma importancia mencionar que para el desarrollo de estos modelos tomaremos en cuenta siete parámetros de entrada, a diferencia de los modelos convencionales que solo utilizan dos parámetros de entrada. Los parámetros que utilizaremos serán (Diferencia en el eje de coordenadas X entre el punto de medición y la voladura, Diferencia en el eje de coordenadas Y entre el punto de medición y la voladura, Diferencia en el eje de coordenadas Z entre el punto de medición y la voladura, Rock Quality Designation (RQD), Unconfined Compressed Strength (UCS), Número de taladros en el proyecto de voladura y La cantidad máxima de explosivo por retardo. El motivo por el cual se decidió utilizar más de dos parámetros de entrada es para lograr modelar de manera más precisa la naturaleza de las vibraciones en el suelo producto de la voladura, considerando todos los parámetros disponibles que a criterio del autor puedan influir en la intensidad de las vibraciones.

Como siguiente paso, se debe realizar una optimización de todos los parámetros requeridos por cada algoritmo para entrenar los distintos modelos predictivos, para lograr esto utilizaremos una librería llamada Hyperopt, la cual configuraremos para utilizar un algoritmo de optimización bayesiano denominado Tree-structured Parzen Estimator Approach (TPE); este algoritmo encontrará los valores óptimos para los parámetros en base a una función objetivo, que en este caso será maximizar los valores de las métricas de desempeño. Así como una optimización de la arquitectura de nuestra red neuronal artificial utilizando la librería

Keras. (Véase Anexo 1: Código de Programación en Lenguaje Python de los Modelos predictivos en base a algoritmos de Machine Learning, Optimización y Aplicación. Y Anexo 2: Código de Programación en Lenguaje Python del del Modelo Predictivo en base a una Red Neuronal Artificial, Optimización.)

Figura 9

Diagrama de Flujo de la metodología adoptada en el presente estudio.



Nota. Elaboración propia

A continuación, procederemos a utilizar nuevamente nuestro IDE, pero esta vez con el objetivo de programar la resolución de las formulaciones empíricas utilizadas comúnmente en la industria minera, siendo las seleccionadas para fines comparativos las siguientes: ecuación USBM por Duvall y Petkof, ecuación Langefors-Kihlstrom, ecuación del Estándar de la India, la ecuación modificada de Ghosh-Daemen y la ecuación Rai and Singh. La finalidad de este paso es obtener las constantes del sitio, requeridas en todas las formulaciones, que minimicen el error en las predicciones de estos modelos.

(Véase Anexo 3: Código de Programación en Lenguaje Python de los Modelos predictivos en base a Formulaciones Empíricas.)

Finalmente, una vez tengamos resultados de cada uno de los modelos, considerando los computacionales y los obtenidos mediante formulaciones empíricas, procederemos a compararlos, se concluirá que un modelo es “mejor” que el resto, si es que obtiene menor error

(RMSE) y un coeficiente de determinación (R2) más cercano a 1, lo que indicaría que ese modelo es capaz de reflejar de mejor manera la variabilidad de los datos. Adicionalmente, se realizará una evaluación de importancia de parámetros, utilizando la librería especializada SHAP, la cual nos permite calcular y graficar la influencia de cada parámetro de entrada en la predicción final de un modelo; para el presente estudio utilizaremos el modelo de Potenciación de Gradiente para determinar la influencia de los parámetros de entrada en las predicciones finales. Una vez concluida la evaluación de resultados, se procederá a dar las conclusiones del estudio, finalizando de esta manera el experimento.

(Véase Anexo 4: Código de Programación en Lenguaje Python del Análisis de Importancia de Parámetros en el modelo de Potenciación de Gradiente.)

3.1. Alcances y Limitaciones

Es prudente y necesario aclarar que la presente investigación no presenta un nuevo diseño de voladura mediante los valores de VPP obtenidos en la experimentación, sino por el contrario busca comparar el desempeño de las formulaciones empíricas a la hora de predecir valores de VPP, con modelos predictivos obtenidos computacionalmente mediante algoritmos de Machine Learning. También resulta importante nombrar y delimitar los algoritmos utilizados en la experimentación, los cuales son: Regresión con Vectores de Soporte, Redes Neuronales Artificiales, Árboles de Decisión, Bosques Aleatorios, Gradient Boosting Machine y Regression Lineal Múltiple. De igual forma se nombrará a continuación aquellas formulaciones empíricas que serán desarrolladas con fines comparativos, siendo estas: ecuación USBM por Duvall y Petkof, ecuación Langefors-Kihlstrom, ecuación del Estándar de la India y la ecuación modificada de Ghosh-Daemen.

De igual manera, es necesario indicar que los modelos a desarrollar en base a algoritmos de Machine Learning son aplicables solamente en los dominios considerados dentro de la base de datos, esto debido a la naturaleza de los algoritmos, que pueden representar de manera general un fenómeno en base a la información que se le presente. Sin embargo, el enfoque y la metodología utilizadas en esta investigación pueden ser replicables con facilidad en cualquier otro dominio geológico, ya sea dentro de la misma operación minera o en cualquier otra.

La realización del proyecto se considera viable, puesto que se cuenta con la base de datos necesaria para realizar la experimentación y con todos los medios necesarios para el mismo fin, ya que, al ser un estudio computacional, solo es necesario un computador funcional, el cuál será proporcionado por cuenta propia. El estudio se llevará a cabo en la provincia de Arequipa, departamento de Arequipa, Perú, y puesto que, la totalidad del estudio puede ser llevado a cabo en condiciones de gabinete, no se necesita un lugar o laboratorio especializado para la realización de este. En cuanto al tiempo, no existen factores alternos que puedan retrasar la ejecución del plan establecido para el proyecto. En cuanto a financiación, al ser un estudio computacional no se incurrirá en gastos considerables más que los gastos necesarios e inevitables como vendrían a ser el consumo eléctrico y el acceso a internet, los cuales serán cubiertos en su totalidad por financiamiento propio. Finalmente, en lo que respecta a limitaciones, no se observan ni se esperan situaciones especiales o no planificadas durante la ejecución del presente proyecto.

3.2. Tipo y Nivel de Investigación

3.2.1. Tipo de Investigación

La presente investigación es de tipo Tecnológica, puesto que busca poner a disposición del lector nuevas alternativas de solución a problemas recurrentes en el pasado.

3.2.2. Nivel de Investigación

La investigación planteada es de nivel Experimental puesto que en la misma se ensayan nuevas soluciones o alternativas a un problema en concreto, que en este caso sería la poca precisión de las formulaciones empíricas para predecir valores de VPP.

3.3. Población y Muestra

La base de datos a utilizar para la realización del presente estudio pertenece a una operación minera a tajo abierto del sur de Perú, la cual por motivos de confidencialidad y legalidad no se me es permitido nombrar y la cuál será denominada como “Operación Minera a Tajo Abierto del sur del Perú” en el desarrollo de la presente investigación. Adicionalmente, resulta prudente detallar que la base de datos fue obtenida durante mi

estadía como practicante profesional en la operación minera a mención. Además de ello, para proteger la confidencialidad de la información, los datos serán multiplicados por un factor constante que no altere la naturaleza de estos pero que si modifique los valores reales de la operación.

La base de datos en mención consta de 200 mediciones de Velocidad Pico Partícula pertenecientes a 121 proyectos de voladura, distribuidos en casi la totalidad de dominios geológicos existentes en la operación minera y realizados en el periodo de tiempo correspondiente entre los meses de Marzo y Mayo del año 2019. Cada medición consta de 14 columnas o parámetros, siendo estos: Nombre del proyecto de voladura, Coordenada X del taladro con mayor carga, Coordenada Y del taladro con mayor carga, Coordenada Z del taladro con mayor carga, Código del geófono utilizado, Coordenada X del punto de medición, Coordenada X del punto de medición, Coordenada X del punto de medición, RQD del macizo rocoso, UCS del macizo rocoso, Número de Taladros, Masa de explosivo en el taladro con mayor carga y Medición de Velocidad Pico Partícula.

Debido a la naturaleza de las técnicas de aprendizaje autónomo o machine learning, resulta necesario utilizar la mayor cantidad de datos posibles durante el proceso de "aprendizaje" o "entrenamiento" de los modelos. Es por ello que no se realizarán técnicas de muestreo en el presente estudio. Sin embargo, la base de datos será tratada de dos distintas maneras. En primera instancia, se realizará una limpieza de esta, garantizando que los datos remanentes no sean datos atípicos o anómalos, producidos por distintas causas como pueden ser errores en la medición o errores de calibración; de esta manera garantizamos que la base de datos refleje de manera correcta la naturaleza del fenómeno que buscamos predecir y que los modelos predictivos puedan "aprender" correctamente de esta información. Posteriormente, la base de datos será dividida en dos secciones, en donde el 80% de los datos serán destinados para el aprendizaje de los modelos predictivos de Machine Learning, y el 20% restante será utilizado con propósitos de validación o prueba de los modelos, es decir, se utilizará para medir que tan precisos son los modelos cuando se les presenta información "nueva".

A continuación, en la **Tabla *2*** se presenta un resumen de la base de datos, considerando los principales parámetros y sus respectivas estadísticas.

Tabla 2

Resumen de la Base de Datos

	RQD	UCS	# Taladros	Masa de explosivo en el taladro con mayor carga (kg)	PPV (m/s)
Cantidad	200,00	200,00	200,00	200,00	200,00
Media	60.58	138.60	40.05	488.55	35.00
Desviación Estándar	23.01	57.30	8.99	205.34	67.05
Mínimo	15.00	40.00	3.00	8.19	0.76
25%	40.00	90.00	37.00	392.01	5.44
50%	60.00	130.00	41.00	541.66	12.00
75%	83.75	180.00	46.00	657.55	35.48
Máximo	100.00	250.00	52.00	824.28	620.00

Nota: Elaboración propia

3.4. Métodos, Técnicas e Instrumentos de Recolección de Datos

Resulta importante aclarar que el desarrollo del presente estudio se realizará utilizando una base de datos histórica. Siendo el método utilizado, una revisión documental de informes y bases de datos. Es por ello que la técnica de recolección de datos adquiere un carácter documental, donde una hoja de cálculo servirá como instrumento en el que se almacenará los datos recolectados y seleccionados para el presente estudio.

Sin embargo, para mayor entendimiento del lector sobre el origen de este tipo de información, a continuación, se detalla el método y la técnica de recolección de datos del cual surge este tipo de bases históricas. El método consiste en un trabajo de campo, en el cual se busca medir la intensidad de las vibraciones durante el proceso de voladura. Para ello, se utiliza un geófono triaxial ubicado a determinada distancia de la voladura. Se debe documentar la ubicación exacta del geófono en base a un sistema de coordenadas UTM, así como, la ubicación exacta del taladro con mayor carga en el proyecto de voladura. Una vez documentadas estas dos ubicaciones, se programa el geófono para que comience a realizar mediciones de las vibraciones en los tres ejes durante un periodo de tiempo que incluya segundos antes de la voladura, el momento exacto de la misma y segundos después. De esta

manera se obtiene mediciones de velocidad de la onda en función del tiempo para cada uno de los tres ejes, y podemos extraer el valor pico o más alto de cada uno estos. Una vez documentada esta información, se puede calcular una resultante general de velocidad pico partícula con métodos matemáticos, como la transformada rápida de Fourier. Además de los datos en mención se debe documentar las consideraciones geológicas del dominio en el que se encuentre el proyecto de voladura. De igual manera, resulta necesario documentar las principales características del proyecto de voladura, como número de taladros detonados y cantidad máxima de explosivo por detonador. Como se ha podido detallar, la técnica de recolección de datos es de carácter documental, en el que se utiliza como instrumentos una base de datos donde se almacenan los datos relevantes.

3.5. Plan de Análisis Estadístico de los Datos

Una vez programados cada uno de los modelos predictivos, ya sean los desarrollados en función a algoritmos de Machine Learning, o los desarrollados en función de formulaciones empíricas. Se procederá a medir su capacidad predictiva en base a la precisión de sus predicciones cuando se les presenta datos completamente nuevos. Para esto utilizaremos una métrica estadística conocida como Raíz del Error Cuadrático Medio, la cual mide el error entre los datos reales y los que han sido predichos por determinado modelo. Se afirma que un modelo es mejor que otro si es que este entrega menor cantidad de error. De la misma manera se medirá la capacidad del modelo en poder representar la variabilidad de la información, es decir que tan bueno es un modelo en función de su variabilidad explicada. Para esto, utilizaremos une métrica estadística denominada Coeficiente de Determinación (R2). Se afirma que un modelo es mejor que otro, si presenta un valor de R2 más cercano a 1.0 que su contrincante. Para el cálculo de estas dos métricas estadísticas, nos apoyaremos en la librería SKLearn, una librería de código abierto para lenguaje de programación Python, la cual nos permitirá estimar estos valores de manera rápida como cualquier otro software estadístico podría hacerlo.

Finalmente, realizado este análisis, podemos cumplir el objetivo principal de la presente investigación: Comparar modelos predictivos de velocidad pico partícula obtenidos con algoritmos de machine learning con modelos predictivos obtenidos mediante formulaciones empíricas, e indicar cuál de ellos es mejor, en términos estadísticos.

CAPÍTULO IV

4. RESULTADOS Y DISCUSIONES

4.1. Análisis e Interpretación de Resultados

Tras haber realizado la experimentación detallada en el Capítulo III de la presente investigación, se obtuvieron los resultados correspondientes a cada una de las etapas propuestas en el modelo metodológico de esta investigación. En primer lugar, se optimizó los modelos predictivos de Machine Learning, buscando los mejores parámetros de configuración para la ejecución de estos y obteniendo como resultado final un listado de hiper-parámetros para cada uno de los modelos, con los cuales se deberán configurar el aprendizaje de estos algoritmos a fin de alcanzar los mejores resultados posibles. Luego, se procedió a obtener los mejores parámetros posibles para la estimación de valores de velocidad pico partícula con modelos tradicionales empíricos, como resultado de esta etapa obtuvimos un listado de constantes para cada una de las formulaciones empíricas que nos servirán para obtener los mejores resultados posibles. Terminada la etapa de optimización de parámetros, se procedió con la creación y evaluación de todos los modelos predictivos, ya sean empíricos o desarrollados en base a algoritmos de Machine Learning; el resultado de esta etapa es un conjunto de métricas de desempeño para cada uno de los modelos. A continuación, se procedió a representar de manera gráfica las predicciones que entregaron cada uno de los modelos predictivos sobre la base de datos de validación; para finalmente desarrollar un estudio de Importancia de Variables, cuyo resultado es un ranking en el que se muestra que variables tienen mayor influencia sobre las predicciones de velocidad pico partícula.

Tras ejecutar el script que optimiza los parámetros de configuración o Hiper-Parámetros de cada uno de los modelos de Machine Learning, obtenemos las siguientes configuraciones. Para el modelo de Regresión Lineal Múltiple se obtuvieron los siguientes hiper – parámetros detallados en la **Tabla *3***, donde se detalla el grado del polinomio que debiese adoptar este modelo, a fin de minimizar los errores de predicción.

Tabla 3

Hiper - parámetros optimizados y utilizados para el modelo de Regresión Lineal Múltiple

Hiper – Parámetro	Valor
Grado del Polinomio	2,00

Nota. Elaboración propia

Para el modelo de Árboles de Decisión o Decision Trees se obtuvieron los hiper – parámetros detallados en la **Tabla *4***. Dentro de estos parámetros resalta la máxima profundidad o 'max_depth' del árbol de decisión, que debiese adoptar el valor de 6 para minimizar el error de las predicciones; y la función que determina la calidad de las separaciones, siendo la óptima 'friedman_mse'.

Tabla 4

Hiper - parámetros optimizados y utilizados para el modelo de Árboles de Decisión

Hiper – Parámetro	Valor
ccp_alpha	1,75E-03
criterion	'friedman_mse'
max_depth	6,00
max_features	'auto'
min_impurity_decrease	401,30
min_samples_leaf	6,40E-03
min_samples_split	4,85E-02
min_weight_fraction_leaf	9,62E-05

Nota. Elaboración propia

Para el modelo de Bosques Aleatorios o Random Forest se obtuvieron los hiper – parámetros detallados en la **Tabla *5***. Dentro de estos parámetros resalta la máxima profundidad o 'max_depth' del árbol de decisión, que debiese adoptar el valor de 15 para minimizar el error de las predicciones; el número de árboles o 'n_estimators' en el bosque aleatorio, siendo el valor óptimo encontrado de 27 árboles; también es importante mencionar el valor de 'min_impurity_decrease' que determinará si un nodo se separará o no si esta separación induce una reducción en la impureza del árbol menos o igual este valor, siendo 0,13 un valor óptimo encontrado.

Tabla 5

Hiper - parámetros optimizados y utilizados para el modelo de Bosques Aleatorios

Hiper – Parámetro	Valor
ccp_alpha	7,57
max_depth	15,00
max_features	None
min_impurity_decrease	0,13
min_samples_leaf	7,13E+06
min_samples_split	7,41E-04
min_weight_fraction_leaf	9,68E-04
n_estimators	27,00

Nota. Elaboración propia

Para el modelo de Potenciación de Gradiente o Gradient Boosting se obtuvieron los hiper – parámetros detallados en la **Tabla *6***. Si bien es cierto, todos los parámetros funcionan en conjunto para minimizar el error, hay algunos de ellos que pueden ser más relevantes de mencionar debido a su popularidad e importancia sobre el modelo; el primero de ellos es la tasa de aprendizaje o 'eta', la cual se estimó como óptima con un valor de 0,87 unidades; luego tenemos el parámetro denominado 'gamma' que determinará el valor mínimo de la función de separación para que una separación tenga lugar, en esta ocasión se estimó un valor óptimo de 217,47; también, tenemos la máxima profundidad o 'max_depth' del árbol de decisión, que debiese adoptar el valor de 35 para minimizar el error de las predicciones; además, tenemos el número de árboles o 'n_estimators' en el bosque aleatorio, siendo el valor óptimo encontrado de 1352 árboles.

Tabla 6

Hiper - parámetros optimizados y utilizados para el modelo de Potenciación de Gradiente

Hiper–Parámetro	Valor
alpha	6,47
base_score	0,50
booster	'gbtree'
colsample_bylevel	0,44
colsample_bynode	0,37
colsample_bytree	0,54
eta	0,87
gamma	217,47
grow_policy	'depthwise'
importance_type	'gain'
interaction_constraints	-
learning_rate	0,87
max_depth	35,00
max_delta_step	0,00
min_child_weight	15,32
missing	nan
monotone_constraints	' ()'
n_estimators	1352,00
n_jobs	-1,00
num_parallel_tree	1,00
reg_alpha	6,47
reg_lambda	3,97
scale_pos_weight	1,00
subsample	0,94
tree_method	'exact'
validate_parameters	1,00

Nota. Elaboración propia

Para el modelo de Máquina de Vectores de Soporte o Support Vector Machine se obtuvieron los hiper – parámetros detallados en la **Tabla *7***. Es importante mencionar que el método de escala de la información seleccionado o 'standardscaler' fue el de escalado estándar o simple. Además, se encontró que el parámetro de regulación óptimo 'C' debiese tener un valor de 495,34 para minimizar el error en las predicciones. Otro parámetro considerable es el grado polinómico o 'degree', el cúal; determinará la complejidad algebraica del modelo.

Tabla 7

Hiper - parámetros optimizados y utilizados para el modelo de Máquina de Vectores de Soporte

Hiper–Parámetro	Valor
standardscaler	StandardScaler()
C	495,34
cache_size	574,32
coef0	23,59
degree	2,00
epsilon	0,07
gamma	'auto'
shrinking	FALSE
tol	0,01

Nota. Elaboración propia

Para el modelo de Redes Neuronales Artificiales o Artificial Neural Network se obtuvieron los hiper – parámetros detallados en la **Tabla *8*** para definir la arquitectura de la red neuronal, es decir, el número de capas y el número de neuronas por capa necesarias en la red neuronal artificial; así como las funciones de activación por utilizarse en cada una de las capas y si es que necesitan considerar un parámetro de Bias o no. Resulta importante mencionar que la capa de entrada tendrá 7 neuronas, puesto que, se utilizarán 7 variables para realizar las predicciones; y que la capa de salida tiene una sola neurona debido a que nuestro resultado esperado de salida es un único valor.

Tabla 8

Hiper - parámetros de arquitectura optimizados y utilizados para el modelo de Redes Neuronales Artificiales

Capa	Número de Unidades	Función de Activación	¿Utilizar Bias?
Entrada	7	-	-
1	70	"tanh"	True
2	208	"selu"	True
3	24	"relu"	True
4	488	"selu"	True
5	442	"relu"	True
6	208	"softplus"	True
7	300	"sigmoid"	True
8	238	"linear"	True
9	8	"linear"	True
Salida	1	"linear"	True

Nota. Elaboración propia

También se obtuvo los parámetros óptimos de optimización, valga la redundancia, necesarios para entrenar la Red Neuronal Artificial, estos se presentan en la **Tabla *9***. En esta tabla se especifica que se utilizará el optimizador ‘Adam’, con sus respectivas tasas de disminución, beta 1 con un valor de 0,04 que determinará que tan rápido decae la función objetivo en el primer momento de optimización; y beta 2 con un valor de 0,27, que determinará que tan rápido decaerá la función objetivo en el segundo momento de optimización. También se estableció la tasa de aprendizaje o ‘learning_rate’ con un valor de 1,36E-03 y el Mínimo Error Cuadrado (mse) como función de minimización.

Tabla 9

Hiper - parámetros de optimización optimizados y utilizados para el modelo de Redes Neuronales Artificiales

Hiper–Parámetro	Valor
optimizer	'Adam'
learning_rate	1,36E-03
beta_1	0,04
beta_2	0,27
epsilon	3,17E-05
amsgrad	False
función de minimización	'mse'

Nota. Elaboración propia

Finalmente, se obtuvo los parámetros óptimos de entrenamiento necesarios para minimizar el error en nuestro modelo de Redes neuronales, los cuales se detallan a continuación en la **Tabla *10***.

Tabla 10

Hiper - parámetros de entrenamiento optimizados y utilizados para el modelo de Redes Neuronales Artificiales

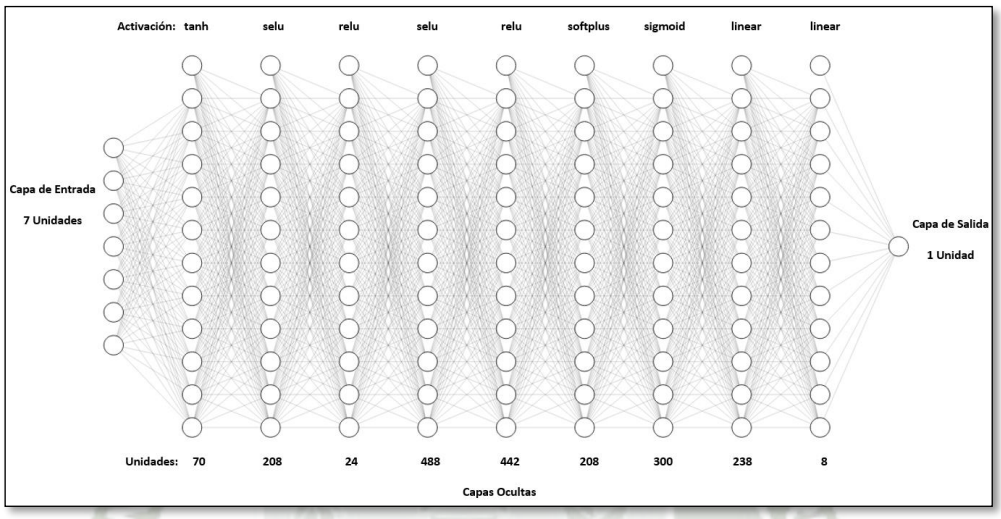
Hiper–Parámetro	Valor
batch_size	50
epochs	3000
EarlyStopping	True
monitor	'val_loss'
patience	600
mode	min'
restore_best_weights	True

Nota. Elaboración propia

Otra forma de visualizar la configuración o arquitectura de nuestra red neuronal es la presentada en la **Figura 10**, donde se observa de manera gráfica y explicita la distribución de las capas y neuronas en la Red Neuronal Propuesta.

Figura 10

Arquitectura de la Red Neuronal Artificial utilizada en el presente estudio



Nota. Elaboración propia

Complementariamente en la **Tabla *11*** se muestran las constantes del sitio optimizadas para cada uno de los modelos empíricos. Con la utilización de estas constantes minimizamos el error en las predicciones.

Tabla 11

Constantes Optimizadas para Modelamientos Empíricos

	K	B	A	alpha
USBM	308,04	1,05	-	-
Langefors	1,03	1,28	-	-
Indian Standard	1,03	2,55	-	-
Ghosh Daemen	551,76	0,72	-	3,02E-03
Rai & Signh	219,84	0,55	0,29	4,42E-03

Nota. Elaboración propia

Una vez que corremos el programa o script que evalúa y compara nuestros modelos de Machine Learning y nuestros modelos empíricos, obtenemos los valores de Coeficiente de Determinación y RMSE para cada uno de los modelos predictivos evaluándose sobre la base de datos de entrenamiento y la base de datos de validación.

En la **Tabla 12** podemos observar que, dentro del grupo de los modelos desarrollados en base a formulaciones empíricas, el que presenta mejores resultados, menor error y mayor coeficiente de determinación, es el modelo Rai & Singh (2004); con un error en la base de datos de validación de 20,65 unidades y un coeficiente de determinación de 0,61. El segundo mejor modelo, en base a la experimentación realizada y a los resultados entregados por esta, es el modelo Gosh Daemen (1983); el cual entregó valores de 20,81 y 0,60 para el error en la base de datos de validación y el coeficiente de determinación, respectivamente. En tercer lugar, tenemos al modelo desarrollado por Duvall y la USBM (1962), entregando un valor de RMSE igual a 21,63 unidades y un coeficiente de determinación igual a 0,57; evaluándose en la base de datos de validación. Los modelos que mostraron el peor desempeño fueron los modelos Indian Standard (1973) y el modelo Langefors – Kihlstrom (1978), entregando ambos un error de 25,21 unidades y un coeficiente de determinación de 0,42; evaluándose ambos en la base de datos de validación.

Los dos últimos modelos mencionados comparten la característica de entregarle mayor peso o prioridad a la carga máxima por retardo que a la distancia entre los puntos de medición y la voladura. Por el contrario, el resto de los modelos prioriza la distancia entre los puntos de intereses sobre la cantidad de explosivo máxima a utilizarse en un taladro. Adicionalmente, los modelos que consideran constantes de elasticidad son aquellos que lograron obtener los mejores resultados en la presente experimentación, Gosh Daemen (1983) y Rai & Singh (2004). Y finalmente, se observa que aquel modelo que asume un comportamiento superior al lineal es el modelo que obtuvo las mejores métricas y por ende el modelo que mejor puede reflejar la naturaleza del fenómeno estudiado, siendo este, el modelo Rai & Singh (2004).

Por otra parte, dentro del grupo de modelos desarrollados en base a algoritmos de Machine Learning, el que presenta mejores resultados es el modelo desarrollado con una Red Neuronal Artificial, entregando un error en la base de datos de validación de 11,54 unidades y un coeficiente de determinación de 0,88. El segundo mejor modelo fue el

modelo con Árboles de Decisión, el cuál entregó un error de 15,79 unidades y un coeficiente de determinación de 0,77; ambas métricas evaluadas en la base de datos de validación. Detrás de este modelo, tenemos al modelo de Potenciación de Gradiente, el cuál al ser evaluado en la base de datos de validación entregó valores de 16,33 y 0,75 para el error alcanzado y el coeficiente de determinación, correspondientemente. El siguiente modelo en el ranking de desempeño, es el modelo de Bosques Aleatorios, entregando métricas de desempeño equivalentes a 18,88 unidades para el error alcanzado y 0,67 para el coeficiente de determinación.

El siguiente modelo, es el de Regresión con Vectores de Soporte, el cuál entregó métricas de 19,25 unidades para el error alcanzado y un coeficiente de determinación de 0,66; ambas métricas obtenidas al evaluarse el modelo en la base de datos de validación. Finalmente, el modelo con el desempeño más bajo en la etapa de validación dentro de este grupo fue el modelo de Regresión Lineal Múltiple, entregando métricas de 19,43 unidades de error alcanzado y un coeficiente de determinación de 0,65 evaluándose en la etapa de validación.

Podemos adjudicar el desempeño óptimo del modelo desarrollado en base a una Red Neuronal Artificial, a la complejidad de la arquitectura planteada en este modelo, al no existir limitantes al momento de intentar modelar un fenómeno con elevado nivel de complejidad, los modelos capaces de encontrar patrones ocultos en la información serán los que mejor desempeño presentarán al evaluarse en este tipo de problemas.

Los tres siguientes modelos en el ranking de desempeño comparten la característica de ser modelos CART (Classification and Regression Trees) o modelos basados en árboles de decisión. Sin embargo, dentro de estos tres modelos, resaltan el modelo básico de Árboles de Decisión y el modelo mejorado con la técnica de Gradient Boosting o Potenciación de Gradiente. El modelo Árboles de Decisión, resalta debido a la poca cantidad de datos considerados en el aprendizaje, ya que es un excelente algoritmo predictivo en situaciones en la que la accesibilidad a los datos es limitada.

Adicionalmente, los modelos de Potenciación de Gradiente y Bosques Aleatorios resaltan debido a que son capaces de reflejar complejas relaciones en los datos, pero sus funcionamientos se ven limitado por la poca cantidad de datos considerada en nuestro estudio. Finalmente, los modelos Máquina de Vectores de Soporte y Regresión Lineal

Múltiple, presentan desempeños no tan aceptables debido a la naturaleza de ambos algoritmos que buscan limitar el modelo a un comportamiento exponencial, sin ser capaz de encontrar patrones ocultos entre la información.

Tabla 12

Resultados de la comparación entre modelos empíricos y modelos de Machine Learning

Modelos	**Datos de Entrenamiento**		**Datos de Validación**	
	Error (RMSE)	Coeficiente de Determinación (R2)	Error (RMSE)	Coeficiente de Determinación (R2)
USBM (1962)	22,32	0,59	21,63	0,57
Indian Standard (1973)	27,71	0,37	25,21	0,42
Langefors – Kihlstrom (1978)	27,71	0,37	25,21	0,42
Ghosh Daemen (1983)	21,29	0,63	20,81	0,60
Rai & Singh (2004)	21,21	0,63	20,65	0,61
Regresión Lineal Múltiple	19,83	0,68	19,43	0,65
Árboles de Decisión	14,96	0,82	15,79	0,77
Bosques Aleatorios	19,65	0,68	18,88	0,67
Potenciación de Gradiente	15,99	0,79	16,33	0,75
Máquina de Vectores de Soporte	19,66	0,68	19,25	0,66
Red Neuronal Artificial	11,65	0,89	11,54	0,88

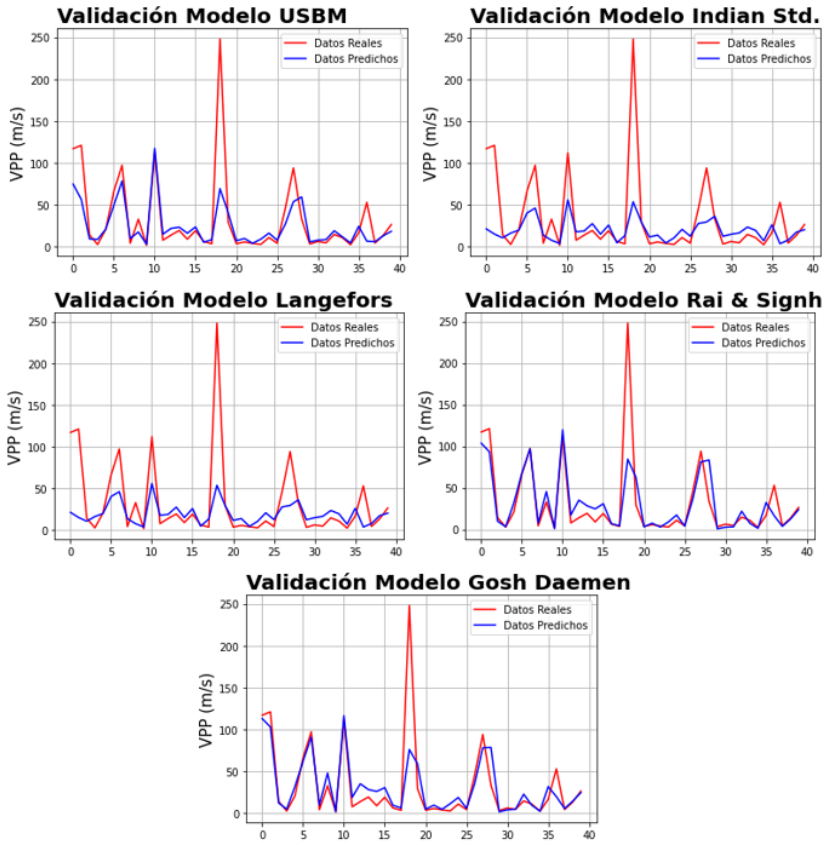
Nota. Elaboración propia

A continuación, se presenta de manera gráfica las predicciones realizadas por cada uno de los modelos desarrollados en base a formulaciones empíricas al ser evaluados en la base de datos de validación.

En la **Figura 11**, observamos de manera gráfica la capacidad de todos los modelos de poder predecir valores totalmente nuevos, como parte del proceso de validación. Resalta en esta evaluación, la vaga capacidad de los modelos desarrollados con formulaciones empíricas para predecir valores altos, es decir, reflejar la variabilidad de la base de datos; esto se comprueba observando los coeficientes de determinación de estos modelos; los cuales, como hemos podido observar en la **Tabla 12**, alcanzan en el mejor modelo un valor de 0,61 y pueden ser tan bajos como 0,42.

Figura 11

Resultados de validación de los modelos empíricos propuestos.

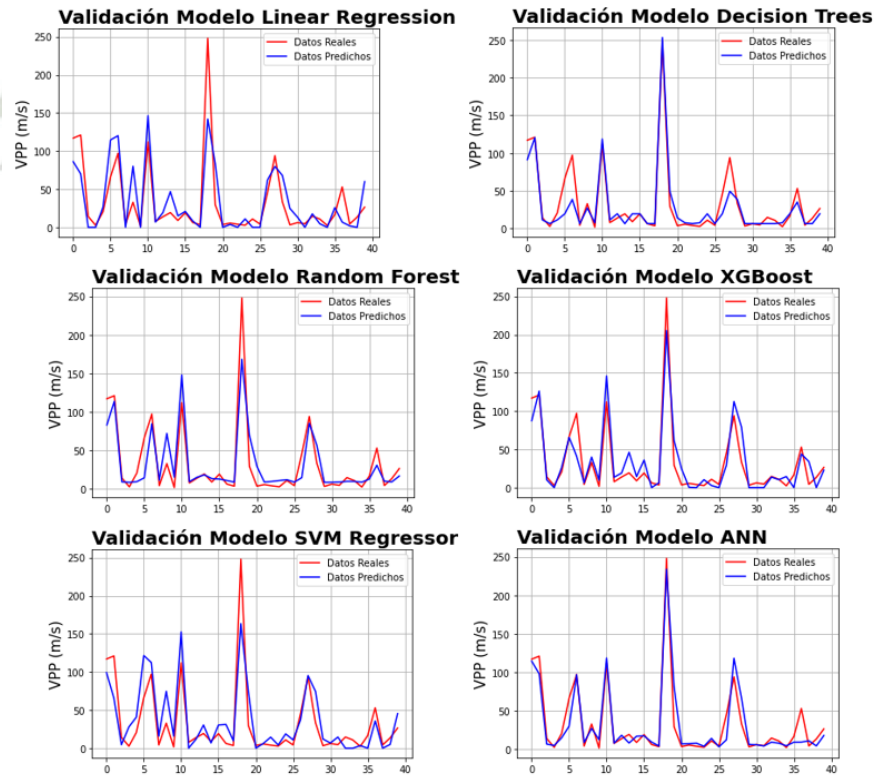


Nota. Elaboración propia

A continuación, se presenta de manera gráfica las predicciones realizadas por cada uno de los modelos desarrollados en base a formulaciones empíricas al ser evaluados en la base de datos de validación. En la **Figura 12**, observamos que la todos los modelos desarrollados con algoritmos de Machine Learning, a excepción del modelo de Regresión Lineal Múltiple, entregan mejores y más acertados resultados a la hora de predecir estos valores críticos. Entre todos los modelos, los que mejor logran predecir los valores de VPP de gran dimensión, son los modelos en base a una Red Neuronal Artificial y el modelo en base a Árboles de Decisión. Sin embargo, podemos observar que el modelo de árboles de decisión "sacrifica" hacer predicciones acertadas en valores inferiores para poder predecir efectivamente los valores más altos; por el contrario, el modelo ANN, gracias a su complejidad interna, logra encontrar un equilibrio entre predecir de la manera más acertada posible los valores altos sin dejar de predecir acertadamente los valores de menor magnitud.

Figura 12

Resultados de validación de los modelos de machine learning

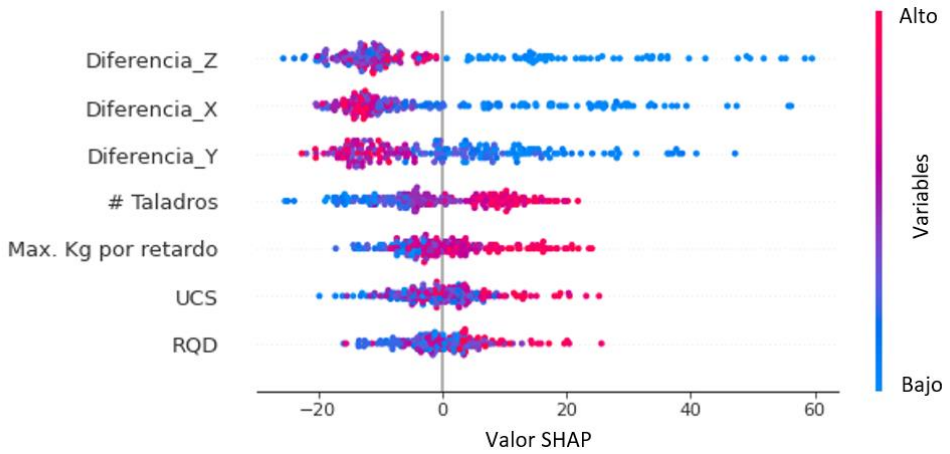


Nota. Elaboración propia

Finalmente, se presentan de manera gráfica los resultados de la evaluación de importancia de parámetros de manera gráfica. Lo cual nos muestra que parámetros tienen mayor impacto sobre la estimación de la Velocidad Pico Partícula. En la **Figura 13** y la **Figura 14**, observamos la influencia de los siete parámetros utilizados en el modelamiento de las predicciones de Velocidad Pico Partícula. En la **Figura 14**, observamos que la diferencia de cotas o diferencia en el eje de coordenadas Z presenta un valor de influencia promedio de aproximadamente 15,80; significando que en la mayoría de los ejemplos, este parámetro influye de manera positiva en la predicción final de los valores de VPP.

Figura 13

Representación gráfica del impacto de las variables sobre el resultado del modelo en todos los datos de entrenamiento evaluados (Valor SHAP)



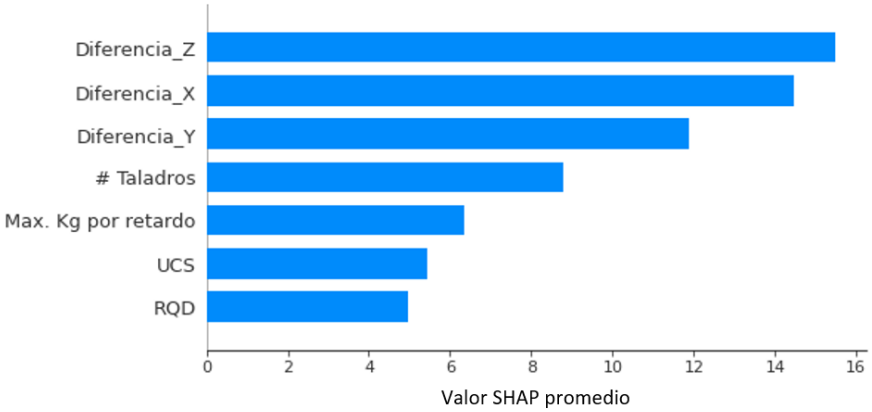
Nota. Elaboración propia

Adicionalmente en la **Figura 13**, observamos que los valores de menor magnitud son los que mayor influencia positiva entregan, es decir, mientras menos diferencia de cotas exista entre los puntos de interés, mayor será el valor de VPP. De igual manera para la diferencia en los ejes X y Y, presentan este mismo comportamiento, pero en menor magnitud; entregando valores de influencia promedios de 14,2 y 11,9; respectivamente.

A continuación, observamos que el valor promedio de influencia para el Número de Taladros en la Voladura es de 8,8; y que, a diferencia de los parámetros mencionados con anterioridad, en la **Figura 14** se observa que un valor de mayor magnitud influirá de manera positiva en la predicción del valor de VPP, es decir que, a mayor cantidad de taladros, más alto será el valor de VPP predicho. De la misma manera, la máxima cantidad de kilogramos por retardo presenta el mismo comportamiento que el número total de taladros, pero con una influencia promedio de 6.5. Finalmente, los factores geomecánicos, RQD y UCS, presentan una influencia promedio de 4,9 y 5,2; respectivamente; al analizar el comportamiento de estos dos parámetros en la **Figura 13**, observamos que valores altos de RQD y UCS, influyen de manera positiva en las predicciones, significando que a mayor RQD o UCS, el valor de VPP será más alto, este comportamiento resulta extraño pero podría explicarse por la existencia de una variable externa no considerada en el estudio, como la elasticidad del macizo rocoso; o por una estimación de factores geomecánicos errónea en la unidad minera.

Figura 14

Representación gráfica del impacto promedio de las variables sobre el resultado del modelo (Valor SHAP promedio)



Nota. Elaboración propia

CONCLUSIONES

- Los modelos desarrollados en base a algoritmos de Machine Learning entregan mejores métricas de desempeño que todos los modelos desarrollados en base a formulaciones empíricas. Entonces, podemos concluir que los modelos en mención representan una mejor opción en la tarea de predecir valores de Velocidad Pico Partícula, en consecuencia, deben ser considerados por las empresas mineras en sus estudios de vibraciones para el diseño de voladuras cercanas a estructuras.

- Dentro de los modelos desarrollados mediante formulaciones empíricas, el que mejores resultados ofrece es desarrollado con la formula empírica presentada por Rai & Singh (2004). Esta superioridad se explica gracias a la no linealidad que ofrece el modelo, asumiendo un comportamiento exponencial, a diferencia de la mayoría de las formulaciones convencionales. Adicionalmente, el hecho de que la formulación considere un factor de elasticidad significa reflejar de mejor manera la naturaleza de las vibraciones a través del macizo rocoso.

- Dentro de los modelos desarrollados mediante algoritmos de Machine Learning, el que mejores resultados ofrece es el desarrollado en base a una Red Neuronal Artificial. Podemos explicar esta superioridad gracias a la complejidad considerada en la arquitectura de la red neuronal; permitiéndole al modelo encontrar patrones ocultos y correlaciones complejas entre los parámetros de entrada, resultando en un modelamiento mucho más complejo y semejante a la naturaleza del fenómeno estudiado.

- Tras realizar el análisis de importancia de parámetros, determinamos que la diferencia de cotas o diferencia de coordenadas en el eje “Z”, influye en mayor magnitud a la hora de estimar un valor de Velocidad Pico Partícula, comparándose con la diferencia con los ejes “X” y “Y”. En consecuencia, concluimos que es necesario desglosar la distancia, entre los dos puntos de interés, en sus tres componentes espaciales para estimar de manera más precisa un valor de VPP. Significando así, que los modelos desarrollados en base a formulaciones empíricas presentan una limitante a la hora de modelar este fenómeno, puesto que estos consideran la resultante de las tres componentes sin priorizar la influencia independiente de cada una.

- De igual manera, en el análisis de importancia de parámetros, se determinó que el número total de taladros presenta un valor de influencia promedio mayor que el perteneciente

a la cantidad máxima de explosivo por taladro. Podemos concluir, que número total de taladros en la voladura es mucho más relevante a la hora de predecir valores de Velocidad Pico Partícula, que la cantidad máxima de explosivo por taladro. En consecuencia, los modelos empíricos no alcanzarán la precisión de los modelos de Machine Learning, por el hecho que no consideran un parámetro que resulta ser sumamente relevante para el modelamiento del comportamiento de las ondas vibratorias en el macizo rocoso.

- Los parámetros geomecánicos considerados en el presente estudio, presentan valores de influencia relativamente bajos y muestran un comportamiento contradictorio a la realidad en el análisis de importancia de parámetros. Puesto que, según el análisis, mientras mejores características geomecánicas presente el macizo, mayor será el valor de VPP. Concluimos que este comportamiento puede ser causado por la existencia de una tercera variable no considerada en la presente experimentación, como podría ser, la elasticidad del macizo. Se realizarán las recomendaciones pertinentes para futuras investigaciones en la siguiente sección.

- Las métricas seleccionadas para evaluar y comparar el desempeño de los modelos desarrollados a lo largo de la realización del presente estudio fueron: El coeficiente de determinación (R2) que nos permite establecer de manera cuantitativa que tanta variabilidad de los datos se puede reflejar en determinado modelo; y la raíz del error medio cuadrado (RMSE), el cual nos permite calificar cuantitativamente que tan preciso o certero es determinado modelo prediciendo valores, en este caso, de Velocidad Pico Partícula. La utilización en paralelo de estas dos métricas nos permite evaluar de manera objetiva el desempeño de los modelos, y determinar cuál de ellos es el más recomendable a la hora de modelar el fenómeno estudiado.

- Durante la realización de un proyecto que considere la implementación de técnicas de Machine Learning, es necesario seleccionar de manera adecuada el algoritmo a utilizar para la obtención de resultados más acertados y de mayor calidad. Al ser este un proyecto comparativo, la utilización de distintos algoritmos, basados en distintas técnicas, resulta ser la mejor estrategia para afrontar la problemática planteada, ya que nos permite experimentar y evaluar distintas alternativas, con la finalidad de escoger la que mejor se adecue a nuestra meta.

- Los algoritmos que buscan ajustarse a una curva, como la Regresión Lineal Múltiple y Regresión con Vectores de Soporte, resultan muy útiles en problemas de regresión como el tratado en este estudio, puesto que buscarán encontrar el modelo que minimice la distancia entre las mediciones y el modelo planteado y, por ende, minimizan el error en las predicciones. Los algoritmos basados en árboles de decisión, como Árboles de Decisión, Bosques Aleatorios y Potenciación de Gradiente, también representan una alternativa viable para solucionar problemas de este tipo, puesto que la flexibilidad en su comportamiento y la nula limitación en su estructura interna nos permite representar modelos altamente complejos, como el necesario en esta investigación. Finalmente, los algoritmos de aprendizaje profundo como las redes neuronales artificiales nos permiten encontrar y representar la complejidad de la información que deseamos modelar; este modelo es uno de los más utilizados en la actualidad debido a su flexibilidad y alto rendimiento, y no debería faltar en un estudio comparativo como el realizado.

- El proceso de selección de parámetros de configuración es parte fundamental en la tarea de implementar modelos de Machine Learning, puesto que la correcta decisión de los valores a utilizar para configurar el aprendizaje mediante estos algoritmos decidirá el desempeño de los modelos. Al ser tratado como un proceso de optimización computacional, limitamos la decisión del programador y reducimos el error provocado por la posible selección de valores no óptimos. De esta manera obtendremos los parámetros más adecuados que nos permitan optimizar nuestra función objetivo, que siempre será reducir el error en las predicciones.

RECOMENDACIONES

- Para todo aquel lector que desee utilizar la presente investigación como base teórica o antecedente para futuras investigaciones, se recomienda considerar cuantos parámetros de entrada estén a su disposición, como podrían ser, parámetros de diseño de voladura, propiedades geo-físicas del macizo rocoso o propiedades de los explosivos utilizados; esto con la finalidad de poder determinar de manera más apropiada cuales son los parámetros con mayor influencia en la generación de vibraciones producto de la voladura. Adicionalmente, se recomienda, trabajar con una base de datos de mayor tamaño. La naturaleza del Machine Learning, radica en trabajar con grandes volúmenes de información y poder generalizar un modelo ante distintas situaciones. De esta manera, sería posible generar un modelo aplicable a toda una unidad minera, y no solo a unas cuantas unidades geológicas como en el presente estudio. Finalmente, se recomienda realizar una optimización de parámetros de configuración cada vez que se busque realizar un estudio de este tipo. A pesar de que se busque estudiar el mismo fenómeno, las realidades difieren de lugar en lugar, y los parámetros encontrados óptimos en este estudio, no serán los mejores al aplicarse en una nueva base de datos que refleje una realidad distinta.
- Se recomienda a todo aquel interesado en las ciencias computacionales aplicadas en la industria minera, utilizar la presente investigación como una motivación para seguir generando conocimiento en esta área. El poder manejar efectivamente los grandes volúmenes de datos que se generan a diario en una operación minera, nos permitirán tomar decisiones óptimas y de acorde a la realidad. Las técnicas de aprendizaje autónomo son aplicables en muchas tareas de la industria, solo hace falta identificar las oportunidades y comenzar a desarrollar las soluciones del futuro.
- Se recomienda a las empresas mineras utilizar modelos predictivos desarrollados mediante algoritmos de Machine Learning, puesto que, en base a los resultados de la presente investigación, representan una mejor alternativa que los modelos predictivos utilizados convencionalmente y desarrollado en función de formulaciones empíricas. De esta manera, lograran predecir de manera más acertada los valores de Velocidad Pico Partícula en sus operaciones, y, en consecuencia, podrán diseñar proyectos de voladura que no generen daño significativo a las estructuras cercanas.

REFERENCIAS BIBLIOGRÁFICAS

Altman, N., & Krzywinski , M. (2015). Association, correlation and causation. *Nature Methods*, 899–900.

Álvarez-Vigil, A., González-Nicieza, C., López Gayarre, F., & Álvarez-Fernández, M. (2012). Predicting blasting propagation velocity and vibration frequency using artificial neural networks. *International Journal of Rock Mechanics and Mining Sciences, 55*, 108-116. doi:https://doi.org/10.1016/j.ijrmms.2012.05.002

Bakhshandeh Amnieh, H., Siamaki, A., & Soltani, S. (2012). Design of blasting pattern in proportion to the peak particle velocity (PPV): Artificial neural networks approach. *Safety Science, 50*(9), 1913-1916. doi:https://doi.org/10.1016/j.ssci.2012.05.008

Biau, G. (2012). Analysis of a Random Forests Model. *Journal of Machine Learning Research*, 1063-1095.

Bramer, M. (2016). *Principles of Data Mining.* London: Springer-Verlag London.

Brownlee, J. (21 de March de 2016). *Machine Learning Algorithms*. Obtenido de Machine Learning Mastery: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

Bureau of Indian Standards. (1973). *Indian standard, Criteria for safety and design of structures subjected to under ground blast.* New Dehli: Bureau of Indian Standards.

C.H. Dowding. (1992). Suggested method for blast vibration monitoring. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts, 29*(2), 145–156. doi:https://doi.org/10.1016/0148-9062(92)92124-U

Chai, T., & Draxler, R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? *Geoscientific Model Development*, 1247-1250.

Christmann, A., & Steinwart, I. (2008). *Support Vector Machines.* New York: Springer.

Dehghani, H., & Ataee-pour, M. (2011). Development of a model to predict peak particle velocity in a blasting operation. *International Journal of Rock Mechanics and Mining Sciences, 48*(1), 51-58. doi:https://doi.org/10.1016/j.ijrmms.2010.08.005

Delgado Ponce, M. (2014). *ESTRATEGIAS EN EL DISEÑO DE PERFORACIÓN Y VOLADURA PARA.* Arequipa: Universidad Nacional de San Agustin.

Dobbin, K., & Simon, R. (2011). Optimally splitting cases for training and testing high dimensional classifiers. *BMC Medical Genomics* , 4-31.

Dr. Dataman. (13 de September de 2019). *Medium*. Obtenido de Towards Data Science: https://towardsdatascience.com/explain-your-model-with-the-shap-values-bc36aac4de3d

Duvall, W., & Fogleson, D. (1962). *Review of Criteria for Estimating Damage to Residences from Blasting Vibration.* Pittsburgh: Bureau of Mines Report of Investigations.

Duvall, W., & Petkof, B. (1959). *Spherical Propagation of Explosion of Generated Strain Pulses in Rocks.* Pittsburgh: Bureau of Mines Report of Investigations.

Fafalios, S., Charonyktakis, P., & Tsamardinos, I. (2020). Gradient Boosting Trees. *Gnosis Data Analysis PC*.

Farmer, I. W. (1968). *Engineering Properties of Rocks.* Londres: E & F.N. SPON Ltd.

Fouladgar, N., Hasanipanah, M., & Bakhshandeh Amnieh, H. (2017). Application of cuckoo search algorithm to estimate peak particle velocity in mine blasting. *Engineering with Computers, 33*, 181–189. doi:https://doi.org/10.1007/s00366-016-0463-0

Ghasemi, E., Kalhori, H., & Bagherpour, R. (2016). A new hybrid ANFIS–PSO model for prediction of peak particle velocity due to bench blasting. *Engineering with Computers, 32*, 607–614. doi:https://doi.org/10.1007/s00366-016-0438-1

Ghosh, A., & Daemem, J. (1983). A Simple New Blast Vibration Predictor (Based On Wave Propagation Laws). *The 24th U.S. Symposium on Rock Mechanics (USRMS).* Texas.

Goodfellow, I., Bengio, Y., & Courvile, A. (2016). *Deep Learning.* Massachusetts: The MIT Press.

Gulati, P., Sharma, A., & Gupta, M. (2016). Theoretical Study of Decision Tree Algorithms to Identify Pivotal Factors for Performance Improvement: A Review. *International Journal of Computer Applications* , 19-25.

Gupta, P. (17 de May de 2017). *Towards Data Science*. Obtenido de Decision Trees in Machine Learning: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2014). *Metodología de la Investigación* (6th ed.). México DF.: McGraw-Hill.

Huang, J., Koopialipoor, M., & Armaghani, D. (2020). A combination of fuzzy Delphi method and hybrid ANN-based systems to forecast ground vibration resulting from blasting. *Sci Rep, 10*, 19397. doi:https://doi.org/10.1038/s41598-020-76569-2

Hurwitz, J., & Kirsch, D. (2018). *Machine Learning for Dummies, IBM Limited Edition.* New Jersey: John Wiley & Sons, Inc.

Jayadeva, Khemchandani, R., & Chandra, S. (2017). *Twin Support Vector Machines.* Ámsterdam: Springer International Publishing.

Jimeno, C., Jimeno, E. L., & Francisc, J. (1995). *Drilling and blasting of Rock* (Vol. 30). Rotterdam: A.A. Balkema.

Kahriman, A. (2002). Analysis of ground vibrations caused by bench blasting at can open-pit lignite mine in Turkey. *Environment Earth Science, 41*, 653–661. doi:https://doi.org/10.1007/s00254-001-0446-2

Khandelwal, M., Armaghani, D., Faradonbeh, R., Yellishetty, M., Abd Majid, M., & Monjezi, M. (2017). Classification and regression tree technique in estimating peak particle velocity caused by blasting. *Engineering with Computers, 33*, 45–53. doi:https://doi.org/10.1007/s00366-016-0455-0

Kishore, V. (2018). *Pro Machine Learning Algorithms.* Pradesh: Apress.

Kotu, V., & Deshpande, B. (2015). Classification. En V. Kotu, & B. Deshpande, *Predictive Analytics and Data Mining* (págs. 63-163). Burlington: Morgan Kaufmann.

Kusrini, & Hartati, S. (2007). IMPLEMENTATION OF C4.5 ALGORITHM TO EVALUATE THE CANCELLATION POSSIBILITY OF NEW STUDENT APPLICANTS AT STMIK AMIKOM YOGYAKARTA. *International Conference on Electrical Engineering and Informatics*, (págs. 623-626). Bandung.

Lai, P. H., Trayer, M., Ramakrishna, S., & Li, Y. (2012). Database Establishment for Machine Learning in NILM. *1st International Non-Intrusive Load Monitoring Workshop.* Pittsburgh.

Langefors, U., & Kihlstrom, B. (1978). *The modern techniques of rock blasting.* New York: John Wiley and Sons, Inc.

Liu, Q., Zhang, Y., & Hu, Z. (2008). Extracting Decision Rules from Sigmoid Kernel. *Advanced Data Mining and Applications*, 294-304.

Lopez Jimeno, E. (1982). Influencia de as propiedades de las Rocas y de los Macizos Rocosos en el Diseño y Resultado de las Voladuras. *Cadernos do Laboratorio Xeolóxico de Laxe: Revista de xeoloxía galega e do hercínico peninsular, 3*, 417-460.

Louppe, G. (2014). *Understanding Random Forest.* Liège: University of Liège.

Lyman, R., & Longnecker, M. (2016). *An Introduction to Statistical Methods & Data Analysis.* Boston: Cengage Learning.

Mason, L., Baxter, J., Bartlett, P. L., & Frean, M. R. (2000). Boosting algorithms as gradient descent. *Advances in neural information processing systems*, 512-518.

Mazzur, M. (17 de March de 2015). *Machine Learning*. Obtenido de A Step by Step Backpropagation Example: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Mechelli, A., & Vieira, S. (2019). *Machine Learning.* Ámsterdam: Elsevier.

Menon, A. (16 de September de 2018). *Linear Regression using Gradient Descent.* Obtenido de Towards Data Science: https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931

Mirjalili, S. (2019). *Evolutionary Algorithms and Neural Network.* Ámsterdam: Springer Publishing Company.

Misra, S., Li, H., & He, J. (2020). *Machine Learning for Subsurface Characterization.* Texas: Gulf Professional Publishing .

Mokfi, T., Shahnazar, A., Bakhshayeshi, I., & Mahmodi, A. (2018). Proposing of a new soft computing-based model to predict peak particle velocity induced by blasting. *Engineering with Computers, 34*, 881–888. doi:https://doi.org/10.1007/s00366-018-0578-6

Nguyen, H., Bui, X.-N., Tran, Q.-H., & Moayedi, H. (2019). Predicting blast-induced peak particle velocity using BGAMs, ANN and SVM: a case study at the Nui Beo open-pit coal mine in Vietnam. *Environ Earth Sci, 78*, 479. doi:https://doi.org/10.1007/s12665-019-8491-x

Nunes da Silva, I., Hernane, D., Andrade, R., Bartocci, L., & Dos Reis, S. (2017). *Artificial Neural Network.* Ámsterdam: Springer International Publishing.

Orellana, J. (12 de November de 2018). *Bookdown*. Obtenido de Árboles de decision y Random Forest: https://bookdown.org/content/2031/

Ostertagova, E. (2012). Modelling Using Polynomial Regression. *Procedia Engineering*, 500–506.

Pfleider, E. P. (1972). *Surface Mining.* New York: The American Institute of Mining, Metallurgical, and Petroleum Engineers Inc.

Qi, J., Du, J., Sabato, M., Ma, X., & Lee, C.-H. (2020). On Mean Absolute Error for Deep Neural Network. *IEEE SIGNAL PROCESSING LETTERS*, 14851489.

Quiroz, C. (2015). Monitoreo de vibraciones por voladuras para controlar posibles daños a estructuras en comunidades cercanas. *Encuentro Tecnología e Investigación* .

Rai, R., & Singh, T. (2004). A new predictor fro ground vibration preiction and its comparison with other predictors. *Indian Journal of Engineering & Materials Sciences*, 178-184.

Raileanu, L., & Stoffel, K. (2004). Theoretical Comparison between the Gini Index and Information Gain Criteria. *Annals of Mathematics and Artificial Intelligence*, 77-93.

Rieuf, E. (11 de February de 2017). *Data Science Central*. Obtenido de Data Science Central: https://www.datasciencecentral.com/profiles/blogs/regression-analysis-how-do-i-interpret-r-squared-and-assess-the

Satapathy, S. K., Dehuri, S., Jagadev, A. K., & Mishra, S. (2016). Introduction. En S. K. Satapathy, S. Dehuri, A. K. Jagadev, & S. Mishra, *EEG Brain Signal Classification for Epileptic Seizure Disorder Detection* (pág. 125). Massachusetts: Academic Press .

Shaltout, N., Elhefnawi, M., Rafea, A., & Moustafa, A. (2014). Information Gain as a Feature Selection Method for the Efficient Classification of Influenza Based on Viral Hosts. *World Congress on Engineering* (págs. 978-988). London: WCE 2014.

Siskind, D., Stagg, M., Kopp, J., & Dowding, C. (1980). *Structure Response and Damage Produced by Ground Vibration from Surface Mine Blasting.* Bureau of Mines. Pittsburgh: Bureau of Mines.

Smola, A., & Vishwanathan, S. (2008). *Introduction to Machine Learning.* Cambridge: The press syndicate of the university of cambridge.

Tan, L. (2015). Chapter 17 - Code Comment Analysis for Improving Software Quality. En C. Bird, T. Menzies, & T. Zimmermann, *The Art and Science of Analyzing Software Data* (págs. 493-517). Burlington: Morgan Kaufmann.

Tosun, A. (2020). Modified Scaled Distance Equation Used for Estimation of Peak Particle Velocity. *J Min Sci, 56*, 388–394. doi: https://doi.org/10.1134/S1062739120036677

Witten, I., Eide, F., & Hall, M. (2011). *Data Mining.* Burlington: Morgan Kaufmann.

Xu, Y., & Goodacre, R. (2018). On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the

Generalization Performance of Supervised Learning. *Journal of Analysis and Testing.* , 249–262.

Yedida, R., & Saha, S. (2019). A novel adaptive learning rate scheduler for deep neural networks. *Foundations of Data Science.*

Yiu, T. (12 de June de 2019). *Towards Data Science*. Obtenido de Understanding Random Forest: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Yiu, T. (2 de June de 2019). *Towards Data Science*. Obtenido de Understanding Neural Networks: https://towardsdatascience.com/understanding-neural-networks-19020b758230

ANEXOS

Anexo 1: Código de Programación en Lenguaje Python de los Modelos predictivos en base a algoritmos de Machine Learning, Optimización y Aplicación.

```
# CARGANDO LAS LIBRERIAS ESPECIALIZADAS AL ENTORNO DE DESARROLLO
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import hyperopt as hp
from hyperopt import tpe, fmin, hp, space_eval, Trials
from hyperopt.pyll.base import scope
from hyperopt.pyll.stochastic import sample
from sklearn.preprocessing import StandardScaler
from hyperopt.early_stop import no_progress_loss
import keras

# CARGANDO LA BASE DE DATOS DESDE UN ARCHIVO .XLSX
file_name = "limpiado"
main_path = ("D:\Desktop\Modelo Predictivo PPV\database")
file_path = (file_name + ".xlsx")
sheet_name = "data"
dataset = pd.read_excel(main_path + "\\" + file_path, sheet_name)
dataset=dataset.sample(n=200,random_state=1)
```

```
# DEFINIENDO LOS PARÁMETROS DE ENTRADA Y LOS DE SALIDA
X=dataset[["Diferencia_X","Diferencia_Y","Diferencia_Z","RQD",
           "UCS","# Taladros","Kg de columna explosiva"]]
y=dataset["PPV"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3,shuffle=True)
X_train, X_test, y_train, y_test = np.array(X_train), np.array(X_test),
np.array(y_train), np.array(y_test)
y_train.shape, y_test.shape = (-1, 1), (-1, 1)

# LISTAS PARA ALMACENAR RESULTADOS
R2_values_train=[]
RMSE_values_train=[]
R2_values_test=[]
RMSE_values_test=[]
index=[]

#############################################################################
######################### REGRESIÓN LINEAL MÚLTIPLE #########################
#############################################################################

########## OPTIMIZADOR ############
# DEFINIENDO LA FUNCIÓN OBJETIVO A OPTIMIZAR
def objective_function_LR(params):

    degree=params["degree"]
    model=make_pipeline(StandardScaler(),
                        PolynomialFeatures(degree),
                        LinearRegression())
    model.fit(X_train,y_train)
    pred_test=model.predict(X_test)
    error= MSE(y_test,pred_test)**(1/2)
```

```
    r2=-r2_score(y_test,model.predict(X_test))

    return float(r2)

# DEFINIENDO EL ESPACIO DE BUSQUEDA DE LA OPTIMIZACIÓN
search_space = {"degree":hp.choice('degree', np.arange(1,10,1, dtype=int))}
max_evals=10

def select_model(space):

    best_regressor = fmin(fn = objective_function_LR,
                          space = space,
                          algo = tpe.suggest,
                          max_evals = max_evals,
                          early_stop_fn=
                          no_progress_loss(iteration_stop_count=200,
                                           percent_increase=0.0))
    print(space_eval(space, best_regressor))

    return space_eval(space, best_regressor)

params_LR=select_model(search_space)

########### MODELO###########
degree=params_LR["degree"]
model_2=make_pipeline(StandardScaler(),PolynomialFeatures(degree),LinearReg
ression())
model_2.fit(X_train,y_train)

# PREDICIENDO VALORES
pred_test=model_2.predict(X_test)
pred_train=model_2.predict(X_train)
```

```
# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test))
RMSE_values_train.append(MSE(y_train,pred_train)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train))
index.append("Multiple Linear Regression")

# GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.figure()
plt.plot(y_test, color = 'red', label = 'Datos Reales')
plt.plot(pred_test, color = 'blue', label = 'Datos Predichos')
plt.title('Validación Modelo Linear
Regression',fontsize=20,fontweight="bold",loc="left")
plt.ylabel("VPP (m/s)",fontsize=15)
plt.legend()
plt.grid(True)
plt.show()


##############################################################################
######################### DECISION TREES REGRESOR ############################
##############################################################################

########## OPTIMIZADOR ############
# DEFINIENDO LA FUNCIÓN OBJETIVO A OPTIMIZAR
def objective_function_DT(params):

    criterion=params["criterion"]
    splitter=params["splitter"]
    max_depth=params["max_depth"]
    min_samples_split=params["min_samples_split"]
    min_samples_leaf=params["min_samples_leaf"]
```

```
    min_weight_fraction_leaf=params["min_weight_fraction_leaf"]
    max_features=params["max_features"]
    random_state=params["random_state"]
    max_leaf_nodes=params["max_leaf_nodes"]
    min_impurity_decrease=params["min_impurity_decrease"]
    ccp_alpha=params["ccp_alpha"]
    model= DecisionTreeRegressor(criterion=criterion,
                                 splitter=splitter,
                                 max_depth=max_depth,
                                 min_samples_split=min_samples_split,
                                 min_samples_leaf=min_samples_leaf,
                                 min_weight_fraction_leaf=
                                 min_weight_fraction_leaf,
                                 max_features=max_features,
                                 random_state=random_state,
                                 max_leaf_nodes=max_leaf_nodes,
                                 min_impurity_decrease=
                                 min_impurity_decrease,
                                 ccp_alpha=ccp_alpha)

    model.fit(X_train,y_train)
    pred_test=model.predict(X_test)
    error= MSE(y_test,pred_test)**(1/2)
    r2=-r2_score(y_test,model.predict(X_test))

    return float(r2)

# DEFINIENDO EL ESPACIO DE BUSQUEDA DE LA OPTIMIZACIÓN
search_space = {"criterion":hp.choice('criterion',
                                      ['mse', 'friedman_mse','mae']),
                "splitter":hp.choice('splitter', ['best', 'random']),
                "max_depth":hp.quniform('max_depth', 1, 10,1),
                "min_samples_split":hp.uniform('min_samples_split', 0, 1),
```

```
                "min_samples_leaf":hp.uniform('min_samples_leaf', 0, 0.5),
                "min_weight_fraction_leaf":hp.uniform(
                                        'min_weight_fraction_leaf', 0, 0.5),
                "max_features":hp.choice(
                                    'max_features', ['auto', 'sqrt','log2',None]),
                "random_state":sample(scope.int(hp.quniform(
                                                'random_state', 1, 7, 1))),
                "max_leaf_nodes":None,
                "min_impurity_decrease":hp.uniform('min_impurity_decrease',
                                                                    0, 500),
                "ccp_alpha":hp.uniform('ccp_alpha', 0, 3)
                }
max_evals=3000

def select_model(space):

    best_regressor = fmin(fn = objective_function_DT,
                          space = space,
                          algo = tpe.suggest,
                          max_evals = max_evals,
                          early_stop_fn=
                          no_progress_loss(iteration_stop_count=1000,
                                           percent_increase=0.0))

    print(space_eval(space, best_regressor))

    return space_eval(space, best_regressor)

params_DT=select_model(search_space)

########### MODELO###########
criterion=params_DT["criterion"]
splitter=params_DT["splitter"]
```

```
max_depth=params_DT["max_depth"]
min_samples_split=params_DT["min_samples_split"]
min_samples_leaf=params_DT["min_samples_leaf"]
min_weight_fraction_leaf=params_DT["min_weight_fraction_leaf"]
max_features=params_DT["max_features"]
random_state=params_DT["random_state"]
max_leaf_nodes=params_DT["max_leaf_nodes"]
min_impurity_decrease=params_DT["min_impurity_decrease"]
ccp_alpha=params_DT["ccp_alpha"]

model_3= DecisionTreeRegressor(criterion=criterion,
                               splitter=splitter,
                               max_depth=max_depth,
                               min_samples_split=min_samples_split,
                               min_samples_leaf=min_samples_leaf,
                               min_weight_fraction_leaf=
                               min_weight_fraction_leaf,
                               max_features=max_features,
                               random_state=random_state,
                               max_leaf_nodes=max_leaf_nodes,
                               min_impurity_decrease=min_impurity_decrease,
                               ccp_alpha=ccp_alpha)
model_3.fit(X_train,y_train)

# PREDICIENDO VALORES
pred_test_3=model_3.predict(X_test)
pred_train_3=model_3.predict(X_train)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_3)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_3))
RMSE_values_train.append(MSE(y_train,pred_train_3)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_3))
```

```
index.append("Decision Trees")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.figure()
plt.plot(y_test, color = 'red', label = 'Datos Reales')
plt.plot(pred_test_3, color = 'blue', label = 'Datos Predichos')
plt.title('Validación Modelo Decision
Trees',fontsize=20,fontweight="bold",loc="left")
plt.ylabel("VPP (m/s)",fontsize=15)
plt.legend()
plt.grid(True)
plt.show()

################################################################################
################################ RANDOM FOREST ################################
################################################################################

########## OPTIMIZADOR ############
# DEFINIENDO LA FUNCIÓN OBJETIVO A OPTIMIZAR
def objective_function_RF(params):
    n_estimators=params["n_estimators"]
    criterion=params["criterion"]
    max_depth=params["max_depth"]
    min_samples_split=params["min_samples_split"]
    min_samples_leaf=params["min_samples_leaf"]
    min_weight_fraction_leaf=params["min_weight_fraction_leaf"]
    max_features=params["max_features"]
    random_state=params["random_state"]
    max_leaf_nodes=params["max_leaf_nodes"]
    min_impurity_decrease=params["min_impurity_decrease"]
    bootstrap=params["bootstrap"]
    oob_score=params["oob_score"]
    n_jobs=params["n_jobs"]
```

```
    warm_start=params["warm_start"]
    ccp_alpha=params["ccp_alpha"]
    max_samples=params["max_samples"]

    model= RandomForestRegressor(n_estimators=n_estimators,
                                 criterion=criterion,
                                 max_depth=max_depth,
                                 min_samples_split=min_samples_split,
                                 min_samples_leaf=min_samples_leaf,
                                 min_weight_fraction_leaf=
                                 min_weight_fraction_leaf,
                                 max_features=max_features,
                                 random_state=random_state,
                                 max_leaf_nodes=max_leaf_nodes,
                                 min_impurity_decrease=
                                 min_impurity_decrease,
                                 bootstrap=bootstrap,
                                 oob_score=oob_score,
                                 n_jobs=n_jobs,
                                 warm_start=warm_start,
                                 ccp_alpha=ccp_alpha,
                                 max_samples=max_samples)

    model.fit(X_train,y_train.ravel())
    pred_test=model.predict(X_test)
    error= MSE(y_test,pred_test)**(1/2)
    r2=-r2_score(y_test,model.predict(X_test))

    return float(r2)
# DEFINIENDO EL ESPACIO DE BUSQUEDA DE LA OPTIMIZACIÓN
search_space = {'n_estimators': scope.int(hp.quniform('n_estimators',
                                                      10, 200, q=1)),
                "criterion":hp.choice('criterion', ['mse','mae']),
```

```
                "max_depth":hp.quniform('max_depth', 1, 100 ,1),
                "min_samples_split":hp.uniform('min_samples_split', 0, 1),
                "min_samples_leaf":hp.uniform('min_samples_leaf', 0, 0.5),
                "min_weight_fraction_leaf":hp.uniform(
                                        'min_weight_fraction_leaf', 0, 0.5),
                "max_features":hp.choice('max_features',
                                         ['auto', 'sqrt','log2',None]),
                "random_state":sample(scope.int(hp.quniform('random_state',
                                                            1, 50, 1))),
                "max_leaf_nodes":None,
                "min_impurity_decrease":hp.uniform(
                                         'min_impurity_decrease',   0, 100),
                "bootstrap":True,
                "oob_score":False,
                "n_jobs":None,
                "warm_start":hp.choice('warm_start', [True,False]),
                "max_samples":None,
                "ccp_alpha":hp.uniform('ccp_alpha', 0, 20)
                }
max_evals=3000

def select_model(space):

    best_regressor = fmin(fn = objective_function_RF,
                          space = space,
                          algo = tpe.suggest,
                          max_evals = max_evals,
                          early_stop_fn=
                          no_progress_loss(iteration_stop_count=500,
                                           percent_increase=0.0))

    print(space_eval(space, best_regressor))
    return space_eval(space, best_regressor)
```

```
params_RF=select_model(search_space)

########### MODELO ##########
n_estimators=params_RF["n_estimators"]
criterion=params_RF["criterion"]
max_depth=params_RF["max_depth"]
min_samples_split=params_RF["min_samples_split"]
min_samples_leaf=params_RF["min_samples_leaf"]
min_weight_fraction_leaf=params_RF["min_weight_fraction_leaf"]
max_features=params_RF["max_features"]
random_state=params_RF["random_state"]
max_leaf_nodes=params_RF["max_leaf_nodes"]
min_impurity_decrease=params_RF["min_impurity_decrease"]
bootstrap=params_RF["bootstrap"]
oob_score=params_RF["oob_score"]
n_jobs=params_RF["n_jobs"]
warm_start=params_RF["warm_start"]
ccp_alpha=params_RF["ccp_alpha"]
max_samples=params_RF["max_samples"]

model_5=RandomForestRegressor(n_estimators=n_estimators,
                              criterion=criterion,
                              max_depth=max_depth,
                              min_samples_split=min_samples_split,
                              min_samples_leaf=min_samples_leaf,
                              min_weight_fraction_leaf=
                              min_weight_fraction_leaf,
                              max_features=max_features,
                              random_state=random_state,
                              max_leaf_nodes=max_leaf_nodes,
                              min_impurity_decrease=min_impurity_decrease,
                              bootstrap=bootstrap,
```

```
                                oob_score=oob_score,
                                n_jobs=n_jobs,
                                warm_start=warm_start,
                                ccp_alpha=ccp_alpha,
                                max_samples=max_samples)
model_5.fit(X_train,y_train.ravel())

# PREDICIENDO VALORES
pred_test_5=model_5.predict(X_test)
pred_train_5=model_5.predict(X_train)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_5)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_5))
RMSE_values_train.append(MSE(y_train,pred_train_5)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_5))
index.append("Random Forest Regressor")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.figure()
plt.plot(y_test, color = 'red', label = 'Datos Reales')
plt.plot(pred_test_5, color = 'blue', label = 'Datos Predichos')
plt.title('Validación Modelo Random
Forest',fontsize=20,fontweight="bold",loc="left")
plt.ylabel("VPP (m/s)",fontsize=15)
plt.legend()
plt.grid(True)
plt.show()

##############################################################################
############################## XGBOOST REGRESSOR ##############################
##############################################################################
```

```
########## OPTIMIZADOR ###########
# DEFINIENDO LA FUNCIÓN OBJETIVO A OPTIMIZAR
def objective_function_XGB(params):

    booster = params['booster']
    eta = params['eta']
    gamma = params['gamma']
    max_depth = int(params['max_depth'])
    n_estimators = int(params['n_estimators'])
    min_child_weight = params['min_child_weight']
    subsample = params['subsample']
    alpha = params['alpha']
    random_state = params['random_state']
    colsample_bytree = params['colsample_bytree']
    colsample_bylevel = params['colsample_bylevel']
    colsample_bynode = params['colsample_bynode']
    reg_lambda = params['reg_lambda']
    grow_policy = params['grow_policy']
    if booster == 'dart':
        sample_type = params['sample_type']
        normalize_type = params['normalize_type']
        rate_drop = params['rate_drop']
        skip_drop = params['skip_drop']
    if booster == 'gbtree':
        model = xgb.XGBRegressor(objective= 'reg:squarederror',
                                 booster = booster,
                                 eta = eta,
                                 gamma = gamma,
                                 max_depth = max_depth,
                                 n_estimators = n_estimators,
                                 min_child_weight = min_child_weight,
                                 subsample = subsample,
                                 alpha = alpha,
```

```
                                random_state = random_state,
                                colsample_bytree = colsample_bytree,
                                colsample_bylevel = colsample_bylevel,
                                grow_policy = grow_policy,
                                colsample_bynode = colsample_bynode,
                                reg_lambda = reg_lambda,
                                n_jobs = -1)

elif booster == 'dart':
    num_round = 50
    model = xgb.XGBRegressor(objective= 'reg:squarederror',
                                booster = booster,
                                eta = eta,
                                gamma = gamma,
                                n_estimators = n_estimators,
                                min_child_weight = min_child_weight,
                                subsample = subsample,
                                alpha = alpha,
                                random_state = random_state,
                                colsample_bytree = colsample_bytree,
                                sample_type = sample_type,
                                normalize_type = normalize_type,
                                rate_drop = rate_drop,
                                skip_drop = skip_drop,
                                colsample_bylevel = colsample_bylevel,
                                grow_policy = grow_policy,
                                colsample_bynode = colsample_bynode,
                                reg_lambda = reg_lambda,
                                n_jobs = -1)

model.fit(X_train,y_train)

if booster == "gbtree":
```

```
        pred_test = model.predict(X_test)
    elif booster == "dart":
        pred_test = model.predict(X_test, ntree_limit = num_round)
    error= MSE(y_test,pred_test)**(1/2)
    r2=-r2_score(y_test,model.predict(X_test))

    return float(r2)

# DEFINIENDO EL ESPACIO DE BUSQUEDA DE LA OPTIMIZACIÓN
search_space = {'booster': hp.choice('booster', ['gbtree',"dart"]),
        'n_estimators': hp.quniform('n_estimators', 50, 3000, 1),
        'eta': hp.uniform('eta', 0, 1),
        'gamma': hp.uniform('gamma', 1, 500),
        'max_depth': hp.quniform('max_depth', 3, 100, 1),
        'min_child_weight': hp.uniform('min_child_weight', 0, 100),
        'random_state': sample(scope.int(hp.quniform('random_state',
                                                     4, 8, 1))),
        'subsample': hp.uniform('subsample', 0, 1),
        'alpha': hp.uniform('alpha', 1, 8),
        'colsample_bytree': hp.uniform('colsample_bytree', 0, 1),
        'sample_type': hp.choice('sample_type', ['uniform', 'weighted']),
        'normalize_type': hp.choice('normalize_type', ['tree', 'forest']),
        'grow_policy': hp.choice('grow_policy', ['depthwise',
                                                 'lossguide']),
        'rate_drop': hp.uniform('rate_drop', 0, 1),
        'skip_drop': hp.uniform('skip_drop', 0, 1),
        'colsample_bylevel':  hp.uniform('colsample_bylevel', 0, 1),
        'colsample_bynode': hp.uniform('colsample_bynode', 0, 1),
        'reg_lambda':  hp.uniform('reg_lambda', 1, 8)}

max_evals=3000

def select_model(space):
```

```
    best_regressor = fmin(fn = objective_function_XGB,
                          space = space,
                          algo = tpe.suggest,
                          max_evals = max_evals,
                          early_stop_fn=
                          no_progress_loss(iteration_stop_count=500,
                                           percent_increase=0.0))

    print(space_eval(space, best_regressor))
    return space_eval(space, best_regressor)

params_XGB=select_model(search_space)

########### MODELO ##########
booster = params_XGB['booster']
eta = params_XGB['eta']
gamma = params_XGB['gamma']
max_depth = int(params_XGB['max_depth'])
n_estimators = int(params_XGB['n_estimators'])
min_child_weight = params_XGB['min_child_weight']
subsample = params_XGB['subsample']
alpha = params_XGB['alpha']
random_state = params_XGB['random_state']
colsample_bytree = params_XGB['colsample_bytree']
colsample_bylevel = params_XGB['colsample_bylevel']
colsample_bynode = params_XGB['colsample_bynode']
reg_lambda = params_XGB['reg_lambda']
grow_policy = params_XGB['grow_policy']
if booster == 'dart':
    sample_type = params_XGB['sample_type']
    normalize_type = params_XGB['normalize_type']
    rate_drop = params_XGB['rate_drop']
```

```
    skip_drop = params_XGB['skip_drop']

if booster == 'gbtree':
    model_4 = xgb.XGBRegressor(objective= 'reg:squarederror',
                               booster = booster,
                               eta = eta,
                               gamma = gamma,
                               max_depth = max_depth,
                               n_estimators = n_estimators,
                               min_child_weight = min_child_weight,
                               subsample = subsample,
                               alpha = alpha,
                               random_state = random_state,
                               colsample_bytree = colsample_bytree,
                               colsample_bylevel = colsample_bylevel,
                               grow_policy = grow_policy,
                               colsample_bynode = colsample_bynode,
                               reg_lambda = reg_lambda,
                               n_jobs = -1)

elif booster == 'dart':
    num_round = 50
    model_4 = xgb.XGBRegressor(objective= 'reg:squarederror',
                               booster = booster,
                               eta = eta,
                               gamma = gamma,
                               max_depth = max_depth,
                               n_estimators = n_estimators,
                               min_child_weight = min_child_weight,
                               subsample = subsample,
                               alpha = alpha,
                               random_state = random_state,
                               colsample_bytree = colsample_bytree,
```

```
                                sample_type = sample_type,
                                normalize_type = normalize_type,
                                rate_drop = rate_drop,
                                skip_drop = skip_drop,
                                colsample_bylevel = colsample_bylevel,
                                grow_policy = grow_policy,
                                colsample_bynode = colsample_bynode,
                                reg_lambda = reg_lambda,
                                n_jobs = -1)

model_4.fit(X_train,y_train)

# PREDICIENDO VALORES
pred_test_4=model_4.predict(X_test)
pred_train_4=model_4.predict(X_train)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_4)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_4))
RMSE_values_train.append(MSE(y_train,pred_train_4)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_4))
index.append("XGBoost Regressor")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.figure()
plt.plot(y_test, color = 'red', label = 'Datos Reales')
plt.plot(pred_test_4, color = 'blue', label = 'Datos Predichos')
plt.title('Validación Modelo
XGBoost',fontsize=20,fontweight="bold",loc="left")
plt.ylabel("VPP (m/s)",fontsize=15)
plt.legend()
plt.grid(True)
plt.show()
```

```
##############################################################################
#################### SUPPORT VECTOR MACHINE REGRESSOR ########################
##############################################################################

########## OPTIMIZADOR ############
# DEFINIENDO LA FUNCIÓN OBJETIVO A OPTIMIZAR
def objective_function_SVM(params):

    kernel=params["kernel"]
    degree=params["degree"]
    gamma=params["gamma"]
    coef0=params["coef0"]
    tol=params["tol"]
    C=params["C"]
    epsilon=params["epsilon"]
    shrinking=params["shrinking"]
    cache_size=params["cache_size"]
    max_iter=params["max_iter"]

    model = make_pipeline(StandardScaler(),SVR(kernel=kernel,
                                               degree=degree,
                                               gamma=gamma,
                                               coef0=coef0,
                                               tol=tol,
                                               C=C,
                                               epsilon=epsilon,
                                               shrinking=shrinking,
                                               cache_size=cache_size,
                                               max_iter=max_iter))

    model.fit(X_train,y_train.ravel())
    pred_test=model.predict(X_test)
```

```
    error= MSE(y_test,pred_test)**(1/2)
    r2=-r2_score(y_test,model.predict(X_test))

    return float(r2)

# DEFINIENDO EL ESPACIO DE BUSQUEDA DE LA OPTIMIZACIÓN
search_space = {"kernel":hp.choice('kernel', ['sigmoid','rbf']),
                "degree":sample(scope.int(hp.quniform('degree', 1, 5, 1))),
                "gamma":hp.choice('gamma', ['auto', 'scale']),
                "coef0":hp.uniform('coef0', 0, 100),
                "tol":hp.uniform("tol",0.000001,0.1),
                "C":hp.uniform('C', 0, 10000),
                "epsilon":hp.uniform('epsilon', 0, 100),
                "shrinking":hp.choice('shrinking', [True,False]),
                "cache_size":hp.uniform('cache_size', 100, 1000),
                "max_iter":-1
                }
max_evals=3000

def select_model(space):
    best_regressor = fmin(fn = objective_function_SVM,
                          space = space,
                          algo = tpe.suggest,
                          max_evals = max_evals,
                          early_stop_fn=
                          no_progress_loss(iteration_stop_count=500,
                                           percent_increase=0.0))

    print(space_eval(space, best_regressor))
    return space_eval(space, best_regressor)

params_SVM=select_model(search_space)
```

```
########### MODELO ##########
kernel=params_SVM["kernel"]
degree=params_SVM["degree"]
gamma=params_SVM["gamma"]
coef0=params_SVM["coef0"]
tol=params_SVM["tol"]
C=params_SVM["C"]
epsilon=params_SVM["epsilon"]
shrinking=params_SVM["shrinking"]
cache_size=params_SVM["cache_size"]
max_iter=params_SVM["max_iter"]

model_6 = make_pipeline(StandardScaler(),SVR(kernel=kernel,
                                             degree=degree,
                                             gamma=gamma,
                                             coef0=coef0,
                                             tol=tol,
                                             C=C,
                                             epsilon=epsilon,
                                             shrinking=shrinking,
                                             cache_size=cache_size,
                                             max_iter=max_iter))


model_6.fit(X_train,y_train.ravel())
# PREDICIENDO VALORES
pred_test_6=model_6.predict(X_test)
pred_train_6=model_6.predict(X_train)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_6)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_6))
RMSE_values_train.append(MSE(y_train,pred_train_6)**(1/2))
```

```
R2_values_train.append(r2_score(y_train,pred_train_6))
index.append("SVM Regressor")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.figure()
plt.plot(y_test, color = 'red', label = 'Datos Reales')
plt.plot(pred_test_6, color = 'blue', label = 'Datos Predichos')
plt.title('Validación Modelo SVM
Regressor',fontsize=20,fontweight="bold",loc="left")
plt.ylabel("VPP (m/s)",fontsize=15)
plt.legend()
plt.grid(True)
plt.show()


################################################################################
######################### ARTIFICIAL NEURAL NETWORK ############################
################################################################################

########## OPTIMIZADOR EN ANEXO 2 ###########

########### MODELO ##########
model_7=keras.models.load_model("D:\Desktop\Modelo Predictivo
PPV\AMM_bestbestmodel")
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# PREDICIENDO VALORES
pred_test_7 = model_7.predict(X_test)
pred_train_7=model_7.predict(X_train)
```

```
# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_7)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_7))
RMSE_values_train.append(MSE(y_train,pred_train_7)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_7))
index.append("ANN")


#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.figure()
plt.plot(y_test, color = 'red', label = 'Datos Reales')
plt.plot(pred_test_7, color = 'blue', label = 'Datos Predichos')
plt.title('Validación Modelo ANN',fontsize=20,fontweight="bold",loc="left")
plt.ylabel("VPP (m/s)",fontsize=15)
plt.legend()
plt.grid(True)
plt.show()


# MOSTRANDO RESULTADOS FINALES
data={'RMSE_Test':RMSE_values_test,
      'R2_test':R2_values_test,
      'RMSE_train':RMSE_values_train,
      R2_train":R2_values_train}
table=pd.DataFrame(data,index=index)
print(table)
```

Anexo 2: Código de Programación en Lenguaje Python del Optimizador del Modelo Predictivo en base a una Red Neuronal Artificial.

```
# CARGANDO LAS LIBRERIAS ESPECIALIZADAS AL ENTORNO DE DESARROLLO
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
from keras.layers import Dense, Activation
from tensorflow import keras
from keras.models import Sequential
from kerastuner.tuners import RandomSearch
from kerastuner.tuners import BayesianOptimization
from kerastuner.engine.hyperparameters import HyperParameters
from keras.callbacks import EarlyStopping
from keras.callbacks import TerminateOnNaN
import kerastuner


# CARGANDO LA BASE DE DATOS DESDE UN ARCHIVO .XLSX
file_name = "limpiado"
main_path = ("D:\Desktop\Modelo Predictivo PPV\database")
file_path = (file_name + ".xlsx")
sheet_name = "data"
dataset = pd.read_excel(main_path + "\\" + file_path, sheet_name)
dataset=dataset.sample(n=200,random_state=1)

# DEFINIENDO LOS PARÁMETROS DE ENTRADA Y LOS DE SALIDA
X=dataset[["Diferencia_X","Diferencia_Y","Diferencia_Z","RQD",
           "UCS","# Taladros","Kg de columna explosiva"]]
```

```
y=dataset["PPV"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3,shuffle=True)
X_train, X_test, y_train, y_test = np.array(X_train), np.array(X_test),
np.array(y_train), np.array(y_test)
y_train.shape, y_test.shape = (-1, 1), (-1, 1)

# LISTAS PARA ALMACENAR RESULTADOS
R2_values_train=[]
RMSE_values_train=[]
R2_values_test=[]
RMSE_values_test=[]
index=[]

# ESCALANDO LOS PARÁMETROS DE ENTRADA PARA FACILITAR LOS CÁLCULOS
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# DEFINIENDO LA FUNCIÓN OBJETIVO A OPTIMIZAR
def build_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('input_units',
                                 min_value=2,
                                 max_value=512,
                                 step=2),
                    activation = hp.Choice("activation",
                                           values=["linear",
                                                   "relu",
                                                   "sigmoid",
                                                   "softmax",
                                                   "softplus",
```

```
                                                "softsign",
                                                "tanh",
                                                "selu",
                                                "elu"]),
                        use_bias=hp.Boolean("use_bias"),
                        input_dim = 7))

    for i in range(hp.Int("n_layers",0,10)):
        model.add(Dense(units=hp.Int(f"Layer_{i}_units",
                                     min_value=2,
                                     max_value=512,
                                     step=2),
                        activation = hp.Choice(f"Layer_{i}_activation",
                                               values=["linear",
                                                       "relu",
                                                       "sigmoid",
                                                       "softmax",
                                                       "softplus",
                                                       "softsign",
                                                       "tanh",
                                                       "selu",
                                                       "elu"]),
                        use_bias=hp.Boolean(f"Layer_{i}_use_bias")))

    model.add(Dense(units = 1,activation='linear'))

    optimizer = hp.Choice("optimizador",values=["adam",
                                                "adamax",
                                                "adadelta",
                                                "adagrad",
                                                "ftrl",
                                                "nadam",
                                                "rmsprop",
```

```
                                                "sgd"])


    if optimizer=="adam":

        model.compile(optimizer =
        keras.optimizers.Adam(learning_rate=hp.Float("learning_rate",
                                                     min_value=1e-3,
                                                     max_value=1e-1,
                                                     sampling="LOG",
                                                     default=1e-3),
                              beta_1=hp.Float("beta_1",
                                              min_value=0,
                                              max_value=1,
                                              step=0.001,
                                              default=0.9),
                              beta_2=hp.Float("beta_2",
                                              min_value=0,
                                              max_value=1,
                                              step=0.001,
                                              default=0.999),
                              epsilon=hp.Float("epsilon",
                                              min_value=1e-9,
                                              max_value=1e-3,
                                              sampling="LOG",
                                              default=1e-7),
                              amsgrad=hp.Boolean("amsgrad",
                                                 default=False),
                              clipvalue=5.0),
        loss = 'mse')

    elif optimizer=="adamax":
```

```
        model.compile(optimizer =
        keras.optimizers.Adamax(learning_rate=hp.Float("learning_rate",
                                                       min_value=1e-3,
                                                       max_value=1e-1,
                                                       sampling="LOG",
                                                       default=1e-3),
                                beta_1=hp.Float("beta_1",
                                                min_value=0,
                                                max_value=1,
                                                step=0.001,
                                                default=0.9),
                                beta_2=hp.Float("beta_2",
                                                min_value=0,
                                                max_value=1,
                                                step=0.001,
                                                default=0.999),
                                epsilon=hp.Float("epsilon",
                                                min_value=1e-9,
                                                max_value=1e-3,
                                                sampling="LOG",
                                                default=1e-7),
                                clipvalue=5.0),
        loss = 'mse')


    elif optimizer=="adadelta":

        model.compile(optimizer =
        keras.optimizers.Adadelta(learning_rate=hp.Float("learning_rate",
                                                         min_value=1e-3,
                                                         max_value=1,
                                                         sampling="LOG",
                                                         default=1),
```

```
                                rho=hp.Float("rho",
                                             min_value=0,
                                             max_value=1,
                                             step=0.001,
                                             default=0.95),
                                epsilon=hp.Float("epsilon",
                                             min_value=1e-9,
                                             max_value=1e-3,
                                             sampling="LOG",
                                             default=1e-7),
                                clipvalue=5.0),
        loss = 'mse')


elif optimizer=="adagrad":

    model.compile(optimizer =
    keras.optimizers.Adagrad(
            learning_rate=hp.Float("learning_rate",
                                   min_value=1e-3,
                                   max_value=1,
                                   sampling="LOG",
                                   default=1),
            initial_accumulator_value=hp.Float("i_a_v",
                                               min_value=0,
                                               max_value=1,
                                               step=0.001,
                                               default=0.1),
            epsilon=hp.Float("epsilon",
                             min_value=1e-9,
                             max_value=1e-3,
                             sampling="LOG",
                             default=1e-7),
```

```
                clipvalue=5.0),
        loss = 'mse')


elif optimizer=="ftrl":

    model.compile(optimizer =
    keras.optimizers.Ftrl(
        learning_rate=hp.Float("learning_rate",
                               min_value=1e-3,
                               max_value=1e-1,
                               sampling="LOG",
                               default=1e-3),
        learning_rate_power=hp.Float("l_r_p",
                                     min_value=-1,
                                     max_value=0,
                                     step=0.001,
                                     default=-0.5),
        initial_accumulator_value=hp.Float("i_a_v",
                        min_value=0,
                        max_value=1,
                        step=0.001,
                        default=0.1),
        l1_regularization_strength=hp.Float("l1_r_s",
                        min_value=0,
                        max_value=1,
                        step=0.001,
                        default=0),
        l2_regularization_strength=hp.Float("l2_r_s",
                        min_value=0,
                        max_value=1,
                        step=0.001,
                        default=0),
```

```
        l2_shrinkage_regularization_strength=hp.Float("l2_s_r_s",
                        min_value=0,
                        max_value=1,
                        step=0.001,
                        default=0),
        clipvalue=5.0),
    loss = 'mse')


elif optimizer=="nadam":

    model.compile(optimizer =
    keras.optimizers.Nadam(learning_rate=hp.Float("learning_rate",
                                                  min_value=1e-3,
                                                  max_value=1e-1,
                                                  sampling="LOG",
                                                  default=1e-3),
                           beta_1=hp.Float("beta_1",
                                           min_value=0,
                                           max_value=1,
                                           step=0.001,
                                           default=0.9),
                           beta_2=hp.Float("beta_2",
                                           min_value=0,
                                           max_value=1,
                                           step=0.001,
                                           default=0.999),
                           epsilon=hp.Float("epsilon",
                                           min_value=1e-9,
                                           max_value=1e-3,
                                           sampling="LOG",
                                           default=1e-7),
                           clipvalue=5.0),
```

```
        loss = 'mse')

    elif optimizer=="rmsprop":

        model.compile(optimizer =
        keras.optimizers.RMSprop(learning_rate=hp.Float("learning_rate",
                                                        min_value=1e-3,
                                                        max_value=1e-1,
                                                        sampling="LOG",
                                                        default=1e-3),
                                 rho=hp.Float("rho",
                                              min_value=0,
                                              max_value=1,
                                              step=0.001,
                                              default=0.9),
                                 momentum=hp.Float("momentum",
                                              min_value=0,
                                              max_value=1,
                                              step=0.001,
                                              default=0),
                                 epsilon=hp.Float("epsilon",
                                              min_value=1e-9,
                                              max_value=1e-3,
                                              sampling="LOG",
                                              default=1e-7),
                                 centered=hp.Boolean("centered",
                                                     default=False),
                                 clipvalue=5.0),
        loss = 'mse')

    elif optimizer=="sgd":

        model.compile(optimizer =
```

```
        keras.optimizers.SGD(learning_rate=hp.Float("learning_rate",
                                                    min_value=1e-5,
                                                    max_value=1,
                                                    sampling="LOG",
                                                    default=1e-2),
                             momentum=hp.Float("momentum",
                                               min_value=0,
                                               max_value=1,
                                               step=0.001,
                                               default=0),
                             nesterov=hp.Boolean("nesterov",
                                                 default=False),
                             clipvalue=5.0),
        loss = 'mse')

    return model


# OPTIMIZANDO LA ARQUITECTURA DE LA RED NEURONAL
tuner=BayesianOptimization(build_model,
                           objective="val_loss",
                           max_trials=20,#20
                           executions_per_trial=1, #10
                           overwrite=True,
                           directory=
                           "D:\Desktop\Modelo Predictivo PPV\KERAS TUNER")

tuner.search(x=X_train,
             y=y_train,
             verbose=1,
             batch_size=160,
             epochs=500, #500
             callbacks=[EarlyStopping(monitor='val_loss',
```

```
                                    patience=2,
                                    mode="min",
                                    restore_best_weights=True,
                                    min_delta=1)],
              validation_data=(X_test, y_test))

# OBTENIENDO Y GUARDANDO EL MODELO QUE MINIMICE EL ERROR
print(tuner.get_best_hyperparameters()[0].values)
model=tuner.get_best_models(num_models=1)[0]
model.save("D:\Desktop\Modelo Predictivo PPV\AMM_bestbestmodel")
```

Anexo 3: Código de Programación en Lenguaje Python de los Modelos predictivos en base a Formulaciones Empíricas

```
# CARGANDO LAS LIBRERIAS ESPECIALIZADAS AL ENTORNO DE DESARROLLO
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score
from scipy.optimize import curve_fit



# CARGANDO LA BASE DE DATOS DESDE UN ARCHIVO .XLSX
file_name = "limpiado"
main_path = ("D:\Desktop\Modelo Predictivo PPV\database")
file_path = (file_name + ".xlsx")
sheet_name = "data"
dataset = pd.read_excel(main_path + "\\" + file_path, sheet_name)
dataset=dataset.sample(n=200,random_state=1)

# DEFINIENDO LOS PARÁMETROS DE ENTRADA Y LOS DE SALIDA
X=dataset[["Diferencia_X","Diferencia_Y","Diferencia_Z","RQD",
           "UCS","# Taladros","Kg de columna explosiva"]]
y=dataset["PPV"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=3,
                                                    shuffle=True)
X_train, X_test, y_train, y_test = np.array(X_train), np.array(X_test),
np.array(y_train), np.array(y_test)

# LISTAS PARA ALMACENAR RESULTADOS
```

```
R2_values_train=[]
RMSE_values_train=[]
R2_values_test=[]
RMSE_values_test=[]
index=[]

# LISTAS PARA ALMACENAR CONSTANTES OPTIMIZADAS
K_values=[]
B_values=[]
A_values=[]
alpha_values=[]
index_2=[]



################################################################################
################################ USBM DUVALL ###################################
################################################################################

# DEFINIENDO FUNCIÓN QUE RESUELVA LA ECUACIÓN
def function_1 ( X_train , K , B ):

    return K*((X_train[:,0]/X_train[:,1]**0.5)**(-B))

# OPTIMIZANDO LAS CONSTANTES PARA MINIMIZAR EL ERROR
popt, pcov = curve_fit(function_1, X_train, y_train)

# RECLAMANDO LAS CONSTANTES ÓPTIMAS
K=popt[0]
B=popt[1]

# PREDICIENDO VALORES
pred_test=function_1(X_test,K,B)
```

```
pred_train=function_1(X_train,K,B)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test))
RMSE_values_train.append(MSE(y_train,pred_train)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train))
index.append("USBM")

# GUARDANDO LAS CONSTANTES
K_values.append(K)
B_values.append(B)
A_values.append("-")
alpha_values.append("-")
index_2.append("USBM")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.plot(y_test, color = 'red', label = 'Real data')
plt.plot(pred_test, color = 'blue', label = 'Predicted data')
plt.title('Prediction USBM')
plt.legend()
plt.show()









################################################################################
########################## LANGEFORS – KIHLSTROM ###############################
```

```
#############################################################################

# DEFINIENDO FUNCIÓN QUE REUELVA LA ECUACIÓN
def function_2( X_train , K , B ):

    return K*(((X_train[:,1]/(X_train[:,0]**(2/3)))**0.5)**(B))

# OPTIMIZANDO LAS CONSTANTES PARA MINIMIZAR EL ERROR
popt, pcov = curve_fit(function_2, X_train, y_train)

# RECLAMANDO LAS CONSTANTES ÓPTIMAS
K_2=popt[0]
B_2=popt[1]

# PREDICIENDO VALORES
pred_test_2=function_2(X_test,K_2,B_2)
pred_train_2=function_2(X_train,K_2,B_2)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_2)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_2))
RMSE_values_train.append(MSE(y_train,pred_train_2)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_2))
index.append("Langefors")

# GUARDANDO LAS CONSTANTES
K_values.append(K_2)
B_values.append(B_2)
A_values.append("-")
alpha_values.append("-")
index_2.append("Langefors")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
```

```
plt.plot(y_test, color = 'red', label = 'Real data')
plt.plot(pred_test_2, color = 'blue', label = 'Predicted data')
plt.title('Prediction Langefors')
plt.legend()
plt.show()
```

```
##############################################################################
############################## INDIAN STANDARD ###############################
##############################################################################

# DEFINIENDO FUNCIÓN QUE REUELVA LA ECUACIÓN
def function_3( X_train , K , B ):

    return K*((X_train[:,1]/(X_train[:,0]**(2/3)))**(B))

# OPTIMIZANDO LAS CONSTANTES PARA MINIMIZAR EL ERROR
popt, pcov = curve_fit(function_3, X_train, y_train)

# RECLAMANDO LAS CONSTANTES ÓPTIMAS
K_3=popt[0]
B_3=popt[1]

# PREDICIENDO VALORES
pred_test_3=function_3(X_test,K_3,B_3)
pred_train_3=function_3(X_train,K_3,B_3)
```

```
# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_3)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_3))
RMSE_values_train.append(MSE(y_train,pred_train_3)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_3))
index.append("Indian Standard")

# GUARDANDO LAS CONSTANTES
K_values.append(K_3)
B_values.append(B_3)
A_values.append("-")
alpha_values.append("-")
index_2.append("Indian Standard")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.plot(y_test, color = 'red', label = 'Real data')
plt.plot(pred_test_3, color = 'blue', label = 'Predicted data')
plt.title('Prediction Indian Standard')
plt.legend()
plt.show()
```

```
##########################################################################
############################## GOSH - DAEMEN ##############################
##########################################################################
```

```
# DEFINIENDO FUNCIÓN QUE REUELVA LA ECUACIÓN
def function_4(X_train , K , B , alpha):

    return K*((X_train[:,0]/X_train[:,1]**(1/3))**(-B))*np.exp(-
alpha*X_train[:,0])

# OPTIMIZANDO LAS CONSTANTES PARA MINIMIZAR EL ERROR
popt, pcov = curve_fit(function_4, X_train, y_train,p0=[400,1,0.001])

# RECLAMANDO LAS CONSTANTES ÓPTIMAS
K_4=popt[0]
B_4=popt[1]
alpha_4=popt[2]

# PREDICIENDO VALORES
pred_test_4=function_4(X_test,K_4,B_4,alpha_4)
pred_train_4=function_4(X_train,K_4,B_4,alpha_4)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_4)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_4))
RMSE_values_train.append(MSE(y_train,pred_train_4)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_4))
index.append("Ghosh Daemen")

# GUARDANDO LAS CONSTANTES
K_values.append(K_4)
B_values.append(B_4)
A_values.append("-")
alpha_values.append(alpha_4)
```

```
index_2.append("Ghosh Daemen")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.plot(y_test, color = 'red', label = 'Real data')
plt.plot(pred_test_4, color = 'blue', label = 'Predicted data')
plt.title('Ghosh Daemen')
plt.legend()
plt.show()




################################################################################
################################## RAI & SINGH ##################################
################################################################################

# DEFINIENDO FUNCIÓN QUE REUELVA LA ECUACIÓN
def function_5( X_train , K , B , A ,  alpha):

    return K*(X_train[:,0]**(-B))*(X_train[:,1]**(A))*np.exp(-
alpha*X_train[:,0])

# OPTIMIZANDO LAS CONSTANTES PARA MINIMIZAR EL ERROR
popt, pcov = curve_fit(function_5, X_train, y_train,p0=[400,1,1,0.001])

# RECLAMANDO LAS CONSTANTES ÓPTIMAS
K_5=popt[0]
B_5=popt[1]
A_5=popt[2]
alpha_5=popt[3]

# PREDICIENDO VALORES
```

```
pred_test_5=function_5(X_test,K_5,B_5,A_5,alpha_5)
pred_train_5=function_5(X_train,K_5,B_5,A_5,alpha_5)

# EVALUANDO EL MODELO Y LAS PREDICCIONES
RMSE_values_test.append(MSE(y_test,pred_test_5)**(1/2))
R2_values_test.append(r2_score(y_test,pred_test_5))
RMSE_values_train.append(MSE(y_train,pred_train_5)**(1/2))
R2_values_train.append(r2_score(y_train,pred_train_5))
index.append("Rai & Signh")

# GUARDANDO LAS CONSTANTES
K_values.append(K_5)
B_values.append(B_5)
A_values.append(A_5)
alpha_values.append(alpha_5)
index_2.append("Rai & Signh")

#GRAFICANDO LOS RESULTADOS DE LA PRUEBA DE VALIDACIÓN
plt.plot(y_test, color = 'red', label = 'Real data')
plt.plot(pred_test_5, color = 'blue', label = 'Predicted data')
plt.title('Rai & Signh')
plt.legend()
plt.show()




# MOSTRANDO RESULTADOS FINALES
data={'RMSE Test':RMSE_values_test,
      'R2 test':R2_values_test,
      'RMSE_train':RMSE_values_train,
      "R2_train":R2_values_train}
```

```
table=pd.DataFrame(data,index=index)
print(table)

# MOSTRANDO TABLA CON CONSTANTES OPTIMIZADAS
data_2={'K':K_values,'B':B_values,'A':A_values,"alpha":alpha_values}
table_2=pd.DataFrame(data_2,index=index_2)
print(table_2)
```

Anexo 4: Código de Programación en Lenguaje Python del Análisis de Importancia de Parámetros en el modelo XGBoost

```
# CARGANDO LAS LIBRERIAS ESPECIALIZADAS AL ENTORNO DE DESARROLLO
import pickle
import pandas as pd
import shap

# CARGANDO LA BASE DE DATOS DESDE UN ARCHIVO .XLSX
file_name = "limpiado"
main_path = ("D:\Desktop\Modelo Predictivo PPV\database")
file_path = (file_name + ".xlsx")
sheet_name = "data"
dataset = pd.read_excel(main_path + "\\" + file_path, sheet_name)
dataset=dataset.sample(n=200,random_state=1)

# DEFINIENDO LOS PARÁMETROS DE ENTRADA Y LOS DE SALIDA
X=dataset[["Diferencia_X","Diferencia_Y","Diferencia_Z","RQD",
           "UCS","# Taladros","Kg de columna explosiva"]]
y=dataset["PPV"]

# CARGANDO EL MODELO XGBOOST
model_4=pickle.load(open('D:\Desktop\Modelo Predictivo PPV\FINAL
MODELS\Xgboost.sav', 'rb'))

# ANÁLISIS DE IMPORTANCIA DE PARÁMETROS
explainer = shap.TreeExplainer(model_4)
shap_values = explainer.shap_values(X)

shap.summary_plot(shap_values, X)
shap.summary_plot(shap_values, X, plot_type="bar")
```