Hector Alexis Caballero

Comparación de algoritmos de Machine Learning para aplicaciones ambientales

2022

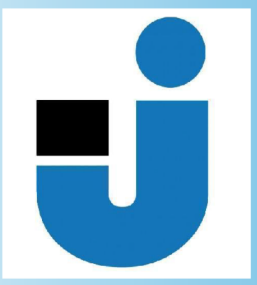
Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

Comparación de algoritmos de Machine Learning para aplicaciones ambientales

Hector Alexis Caballero

Florencio Varela, Agosto 2022

Resumen

El presente informe es un trabajo de investigación enfocado en las bases de Machine Learning y su aplicación para resolver dos ejercicios de predicción. El primero consiste en calcular cuánto cobrará una aseguradora basado en los datos personales del individuo; el segundo consiste en clasificar un celular en un rango de precio dependiendo de sus especificaciones técnicas.

Los algoritmos utilizados para la resolución de la tarea pertenecen a tres tipos de modelos: de Regresión, de Clasificación y de Redes Neuronales Artificiales. En total se utilizaron un total de cinco tipos de algoritmos: Regresión lineal múltiple, K-NN, Árbol de decisión, Bosques aleatorios, y Redes Neuronales Artificiales.

Además, se demostró cómo analizar y preprocesar los datos para aumentar significativamente el rendimiento de los algoritmos.

El resultado de la investigación retornó en todos los modelos un porcentaje de éxito que oscila entre el 79% y 96%.

El informe tiene como principal función actuar como guía e introducción para los estudiantes e interesados en adentrarse al mundo de Machine Learning, tanto de manera teórica como práctica utilizando casos del mundo real.

Abstract

This report is a research work focused on the foundations of Machine Learning and its application to solve two prediction exercises. The first consists of calculating how much an insurer would charge based on the individual's personal data; the second consists of classifying a cellphone in a price range depending on its technical specifications.

The algorithms used to solve the task belong to three types of models: Regression, Classification and Artificial Neural Networks. A total of five types of algorithms were used: Multiple Linear Regression, K-NN, Decision Tree, Random Forests, and Artificial Neural Networks.

In addition, it was demonstrated how to analyze and preprocess the data to significantly increase the performance of the algorithms.

The result of the investigation returned in all the models a percentage of success that oscillates between 79% and 96%.

The report is helpful as a guide and introduction for students and those interested in entering the world of Machine Learning, both theoretically and practically using real-world cases.

1. Introducción

La presente Práctica Profesional Supervisada (PPS) se desarrolló en el marco del Proyecto de Investigación de la Universidad Nacional Arturo Jauretche UNAJ INVESTIGA 2020 (Código del Proyecto 80020200100030UJ y Resolución Rectoral N° 183/21 de fecha 08/08/2021), cuyo título es "Algoritmos de Machine Learning para procesamiento de imágenes en aplicaciones biomédicas, agronómicas y ambientales", bajo la Dirección del Dr. Ing. Martín Morales.

El concepto de Inteligencia Artificial (IA) se aplica a cualquier técnica, centrada en el desarrollo de software o máquinas, que exhiben una inteligencia humana. Dentro de los principales objetivos de la IA se incluyen la deducción y el razonamiento, la representación del conocimiento, la planificación, el procesamiento del lenguaje natural, el aprendizaje, la percepción y la capacidad de manipular y mover objetos.

Para satisfacer dichos objetivos, la IA abarca diferentes áreas de estudio, tales como el Aprendizaje Automático, la Visión por Computadora y la Robótica.

En particular, el Aprendizaje Automático (Machine Learning) es un subcampo de la IA en el que se utilizan diferentes algoritmos que son capaces de aprender de su entorno y adaptarse a nuevas circunstancias, a partir de un conjunto de datos que el algoritmo recibe en la etapa de entrenamiento.

El "aprendizaje" de las computadoras se refiere a la capacidad para detectar e identificar patrones en grandes conjuntos de datos y, a través de ellos, extrapolar, optimizar, tomar decisiones, o hacer una predicción acerca de comportamientos futuros de una situación utilizando un análisis estadístico. De esta manera se permitirá resolver problemas de forma intuitiva y automatizada sin que el mecanismo de elección se encuentre previamente programado. En la práctica, esto se traduce en una función matemática en la que se parte de una entrada y se obtiene una salida; el desafío reside en construir un modelado automático de esta función matemática.

Las características principales de los algoritmos de Machine Learning es que son capaces de resolver problemas no lineales, aprender de ejemplos, encontrar correlaciones entre datos diversos, manejar diferentes tipos de datos (numéricos, textuales, imágenes, etc.), tratar con grandes conjuntos de datos y/o conjuntos de datos de alta dimensión, ser tolerantes

a fallos. Es decir, sobrellevar el ruido y datos incompletos o atípicos, realizar predicciones y generalizaciones a altas velocidades y realizar procesamiento en tiempo real.

Si bien las primeras teorías sobre Machine Learning se desarrollaron en la década del 80, existen dos razones principales por las que solo han empezado a resultar útiles recientemente: una porque requieren de grandes cantidades de datos y otra, además, de una potencia de cálculo significativa.

Las técnicas de Machine Learning son aplicadas en diferentes áreas como: medicina, seguridad, turismo, finanzas, robótica, entre otras. Por ejemplo, para el control de vehículos autónomos, detección de rostros, detección de matrículas, diagnóstico de enfermedades, realidad aumentada, etc.

Con ellas se están consiguiendo, en los campos mencionados, resultados que antes no era posible obtener con los métodos tradicionales. Sin embargo, no existe una única técnica óptima para resolver todos los problemas que se pueden presentar(predicción, reconocimiento facial, calificación, etc.). Cada caso debe analizarse por separado y - de acuerdo con los requisitos del problema- se debe aplicar la técnica adecuada.

Se distinguen tres tipos principales de técnicas de Machine Learning: las de aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

En el primero, los modelos son entrenados a partir de un conjunto de datos en el que la respuesta correcta es conocida. Es decir, se presenta a la red un conjunto de patrones de entrada junto con la salida esperada, y los pesos se van modificando de manera proporcional al error que se produce entre la salida real de la red y la salida esperada.

Por su parte, en el aprendizaje no supervisado, el conjunto de datos empleados en el entrenamiento no contiene la respuesta, de modo que no existe un resultado a reproducir. En este caso, los modelos se ajustan a las observaciones dadas y el objetivo que se persigue es el de modelizar la distribución de los datos para conocer más sobre ellos. Por ejemplo, para determinar posibles patrones ocultos o para agrupar datos de acuerdo con un cierto criterio.

Por último, en el aprendizaje por refuerzo, el algoritmo de aprendizaje recibe algún tipo de valoración (recompensa) acerca de la idoneidad de la respuesta dada. El objetivo en este caso es el de extraer qué acciones deben ser elegidas en los diferentes estados para maximizar la recompensa en base a experiencias pasadas.

El presente trabajo se enfoca en algoritmos de aprendizaje supervisados, los cuales se dividen a su vez en algoritmos de regresión y de clasificación, dependiendo del tipo de resultado que se espera obtener.

Los modelos de regresión son modelos matemáticos que buscan determinar la relación entre una variable dependiente con respecto a otra u otras variables, llamadas *explicativas* o *independientes*. En estos modelos, el resultado es un valor numérico dentro de un conjunto infinito de posibles resultados. Entre las técnicas de Machine Learning que se utilizan en problemas de regresión se destacan: regresión lineal y regresión no lineal, máquinas de vectores de soporte, árboles de decisión, bosques aleatorios y redes neuronales.

Por su parte, en los modelos de clasificación, el resultado es la categoría entre un número limitado de clases de acuerdo al problema específico (por ejemplo: si-no; mal-bien; chico-mediano-grande; etc.). Entre las técnicas de Machine Learning que se utilizan en los problemas de clasificación se encuentran: regresión logística, máquinas de vectores de soporte, árboles de decisión, bosques aleatorios y redes neuronales.

Si bien hay las que son específicas de regresión y otras, de clasificación; la mayoría funcionan para ambos modelos.

1.1 Objetivos de la práctica profesional supervisada

El objetivo de la PPS es la investigación y aplicación de Machine Learning en problemáticas de predicción y clasificación mediante dos ejercicios:

- Clasificación: A partir de los datos recolectados sobre las especificaciones de los celulares, se clasifica el precio de los celulares en base a su información técnica.
- Predicción: Con la combinación de los datos de personas y el precio que pagan en sus seguros, se genera un agente capaz de predecir el precio que se le cobrará a una persona dependiendo de sus datos personales.

1.2 Estructura del trabajo

La PPS se divide en un total de seis secciones divididas de la siguiente forma:

Conceptos generales:

Explicación de la base del Machine Learning para una mejor comprensión de los estudios realizados en el trabajo. Estos integran las distintas definiciones de la IA, características y ramas; qué es Machine Learning, cómo se diferencia del resto de las ramas de la IA, las técnicas de aprendizaje (supervisado, no supervisado y por refuerzo) y las formas de modelo (Clasificación y Regresión/Predicción).

Modelos de Regresión y Clasificación:

Análisis de los siguientes modelos, sus características y diferencias:

- Modelos de Regresión.
  - Lineal
  - Logística
  - Redes neuronales
  - Árboles de Decisión
- Modelos de Clasificación.
  - Redes neuronales
  - Árboles de Decisión.

Herramientas de trabajo:

Se explica el lenguaje implementado, Python, y por qué de su elección así como su ventaja sobre otros lenguajes en el campo de la Inteligencia Artificial. Además, se describen los softwares con los que se desarrolló: Anaconda y Spider. Y por último cómo se preparó el entorno de trabajo de manera local.

Aplicación:

Resolución de los dos ejercicios de clasificación y regresión.

En ambos casos, el procedimiento es el mismo. Primero se analizan los datos, se separan los datos relevantes de los irrelevantes. Luego se ejecutan los algoritmos adaptados a esta optimización para obtener el resultado de cada uno.

2. Temas de estudio, análisis e investigación

2.1 ¿Qué es IA?

El concepto de Inteligencia Artificial es más complejo de lo que se puede llegar a pensar en un primer momento. Se la puede definir como el estudio de la informática tanto en el desarrollo de software como de máquinas capaces de exhibir inteligencia humana o equivalente. Para ello, debe ser capaz de aprender y razonar en base al conocimiento adquirido en su etapa de entrenamiento.

2.1.1 Objetivos de la IA

La razón de la ambigüedad a la hora de su definición se puede distinguir al analizar sus objetivos. Estos engloban la deducción y el razonamiento (adquirido en base al aprendizaje) la representación del conocimiento, la planificación, el procesamiento del lenguaje natural (NLP), el aprendizaje, la percepción y la capacidad de manipular y mover objetos.
A largo plazo, se incluye el logro de la creatividad, la inteligencia social y la inteligencia general (a nivel humano).

2.1.2 Fundamentos de la Inteligencia Artificial

Existen varias disciplinas que han contribuido al campo del desarrollo de la IA con ideas, puntos de vista y técnicas. Es importante hacer un repaso sobre éstas para comprender la complejidad en su definición.

Matemáticas

Para entender y desarrollar los principales algoritmos que se utilizan en el campo de la Inteligencia Artificial, se debe tener noción de:

- Álgebra lineal: Es una rama de las matemáticas que estudia conceptos tales como vectores, matrices, tensores, sistemas de ecuaciones lineales y en su enfoque de manera más formal, espacios vectoriales y sus transformaciones lineales. Una buena

comprensión del álgebra lineal es esencial para entender y trabajar con muchos algoritmos de Machine Learning, y especialmente para los de Deep Learning.

- Cálculo: Es el campo que incluye el estudio de los límites, derivadas, integrales y series infinitas, y más concretamente, es el estudio del cambio. Particularmente para el campo de la IA algunos conceptos relevantes son: Cálculo Diferencial e Integral, Derivadas Parciales, Funciones de Valores Vectoriales, y Gradientes.
- Optimización matemática: Es la herramienta matemática que permite optimizar decisiones, es decir, seleccionar la mejor alternativa de un conjunto de criterios disponibles. Su comprensión es fundamental para poder entender la eficiencia computacional y la escalabilidad de los principales algoritmos de Machine Learning, los cuales suelen trabajar con matrices dispersas de gran tamaño.
- Probabilidad y estadística: Es la rama de la matemática que trata con la incertidumbre, la aleatoriedad y la inferencia. Sus conceptos son fundamentales para cualquier algoritmo de Machine Learning o Deep Learning.

Neurociencias

Es el estudio del sistema neurológico, en especial del cerebro. Uno de sus objetivos es analizar la forma exacta en la que se genera el pensamiento, uno de los grandes misterios de la ciencia. El hecho de que una colección de simples células puede llegar a generar razonamiento, acción, y conciencia es un enigma a resolver. Muchos modelos de IA fueron inspirados en la estructura y el funcionamiento de nuestro cerebro.

Psicología

Es una disciplina que trata sobre el estudio y análisis de la conducta y los procesos mentales de los individuos y grupos humanos. La rama que más influencia ha tenido para la IA es la de la psicología cognitiva que se encarga de los procesos mentales implicados en el conocimiento. Tiene como objeto de estudio los mecanismos básicos y profundos por los que se elabora el conocimiento, desde la percepción, la memoria y el aprendizaje, hasta la formación de conceptos y razonamiento lógico. Las teoría descritas por esta rama han sido utilizados para desarrollar varios modelos de Inteligencia Artificial y Machine Learning.

2.1.3 Enfoques

Se distinguen cuatro enfoques distintos para abordar la definición de la IA y sus objetivos:

- Sistemas que se comportan como un humano: es la idea de desarrollar máquinas o sistemas capaces de realizar funciones para las cuales se necesitaría de un ser humano inteligente. El famoso test de Turing tiene como objetivo poner a prueba a las máquinas y poder distinguir sus comportamientos del de un ser humano real. Para que las máquinas puedan superar ésta prueba, deben, por lo menos, contar con las siguientes capacidades.
    - Procesamiento de lenguaje natural, que le permita comunicarse satisfactoriamente.
    - Representación del conocimiento, para almacenar lo que se conoce o se siente.
    - Razonamiento automático, para utilizar la información almacenada para responder a preguntas y extraer nuevas conclusiones.
    - Aprendizaje automático, para adaptarse a nuevas circunstancias y para detectar y extrapolar patrones.
    - Visión computacional, para percibir objetos.
    - Robótica, para manipular y mover objetos.

- Sistemas que piensan como humanos: en este caso, la idea es lograr que las máquinas tengan las mismas capacidades cognitivas de toma de decisiones, resolución de problemas, aprendizaje, entre otros. Dentro de este enfoque se encuentra el campo interdisciplinario de la ciencia cognitiva, en el cual convergen modelos computacionales de IA y técnicas experimentales de psicología intentando elaborar teorías precisas y verificables sobre el funcionamiento de la mente humana.

- Sistemas que piensan racionalmente: aquí la idea es descubrir los cálculos que hacen posible percibir, razonar y actuar; es decir, encontrar las leyes que rigen el pensamiento racional. Dentro de este enfoque se encuentra la Lógica, que intenta expresar las leyes que gobiernan la manera de operar de la mente.

- Sistemas que se comportan racionalmente: dentro de este enfoque un agente racional es aquel que actúa con la intención de alcanzar el mejor resultado o, cuando hay incertidumbre, el mejor resultado esperado. Un elemento importante a tener en cuenta es que tarde o temprano uno se dará cuenta de que obtener una racionalidad perfecta (hacer siempre lo correcto) no es del todo posible en entornos complejos. Como lo que se busca en este enfoque es realizar inferencias correctas, se necesitan las mismas habilidades que para la Prueba de Turing, es decir, contar con la capacidad para representar el conocimiento y razonar basándose en él, porque ello permitirá alcanzar decisiones correctas en una amplia gama de situaciones. Es necesario ser capaz de generar sentencias comprensibles en lenguaje natural, ya que el enunciado de tales oraciones permite a los agentes desenvolverse en una sociedad compleja. El aprendizaje no se lleva a cabo por erudición exclusivamente, sino que profundizar en el conocimiento de cómo funciona el mundo facilita la concepción de estrategias mejores para manejarse en él.

2.2 ¿Qué es Machine Learning?

Es el diseño y estudio de las herramientas informáticas que utilizan la experiencia pasada para tomar decisiones futuras; es el estudio de programas que pueden aprender de los datos. Su objetivo fundamental es generalizar, o inducir, una regla desconocida a partir de ejemplos donde esa regla es aplicada con el objetivo de dotar a la máquina de la posibilidad de aprender.

El ejemplo más típico donde podemos ver el uso del Machine Learning es en el filtrado de los correos basura o spam. Mediante la observación de miles de correos electrónicos que han sido marcados previamente como basura, los filtros de spam aprenden a clasificar los mensajes nuevos.

Para abordar sus diversas aplicaciones descritas anteriormente, es crucial en primer lugar distinguir los distintos tipos de problemas de Machine Learning con los que uno se puede encontrar:

2.2.1 Aprendizaje supervisado

En esta categoría se enseña o entrena al algoritmo a partir de datos que ya vienen etiquetados con la respuesta correcta. Ésta se define como el valor que se busca predecir mediante el algoritmo. Dicho valor puede ser tanto cuantitativo (como el salario de un empleado o la temperatura de un determinado día), así como calificativo (como cuál es el color de la imagen, o si lloverá o no).

El conjunto de datos supervisado, para su proceso, se divide en dos grupos: entrenamiento y test. El primero es utilizado para estimar los parámetros del modelo y el segundo se emplea para comprobar el comportamiento del modelo estimado en base al aprendizaje obtenido en la fase de entrenamiento.

Existen dos tipos principales de aprendizaje supervisado: clasificación y regresión. En uno se entrena a un algoritmo para clasificar los datos de entrada en variables discretas; mientras que en el otro se entrena el algoritmo para predecir una salida a partir de un rango contínuo de valores posibles.

2.2.2 Aprendizaje no supervisado

Aquí el algoritmo es entrenado usando un conjunto de datos que no tiene ninguna etiqueta; en este caso, nunca se le dice al algoritmo aquello que representan los datos. La idea es que pueda encontrar por sí solo patrones que ayuden a entender el conjunto de datos.

En este tipo de aprendizaje, la máquina es incapaz de etiquetar elementos al no tener ninguna base para ello. Sin embargo, puede agrupar los datos según sus propiedades. Por ejemplo, en caso de imágenes de frutas, podría agruparlas por color, forma y tamaño aún sin saber qué es cada fruta. De esta forma, la máquina agrupará los elementos encontrando similitudes y patrones ocultos.

Al no ser supervisado, no existen respuestas correctas ni incorrectas para el modelo, por lo que su principal función es el análisis de datos. Es por ello que su principal función es la de explorar datos desconocidos, para revelar patrones que una persona es incapaz de detectar debido al volumen de información. Estas estructuras ocultas se denominan vectores de características.

El aprendizaje no supervisado utiliza una técnica llamada *reducción de dimensionalidad*, lo cual ocurre cuando la máquina asume que muchos datos son redundantes

y elimina dimensiones o combina algunas partes de datos según corresponda. La compresión de datos da como resultado ahorro de tiempo y ahorro en potencia de procesamiento.

El aprendizaje no supervisado utiliza una variedad de algoritmos para ajustar los datos en grupos amplios, clústeres y asociaciones como por ejemplo agrupación en clústeres jerárquica, agrupación en clústeres K-means y modelos de mezcla gaussiana, entre otros.

2.2.3 Aprendizaje por refuerzo

Es un área del aprendizaje automático donde la máquina aprende por sí misma el comportamiento a seguir en base a un sistema de recompensas y penalizaciones.

En estos casos, siempre existen los siguientes elementos:

- Un agente, la máquina que debe aprender.
- Las acciones que el agente puede realizar y que lo llevan de un estado a otro.
- Un ambiente definido por estados.
- Recompensas y penalizaciones

A diferencia del aprendizaje supervisado y no supervisado, en esta categoría no se requieren datos históricos sino que sus datos de entrada, recompensa y penalización, provienen de un ambiente definido en base a las acciones del agente.Este aprendizaje se puede dividir en dos etapas: exploración y explotación.

En la etapa de exploración, el agente toma acciones de manera aleatoria para explorar el ambiente y descubrir qué acciones son las más eficaces en determinados estados. En la de explotación, el agente toma decisiones en base a cuán valiosa es dicha decisión en base al estado actual en el que se encuentra. Estas etapas pueden dividirse en espacio de tiempo, configurando el sistema para que explore el 30% del tiempo y explote el 70% restante pues es un sistema que aprende en tiempo real.

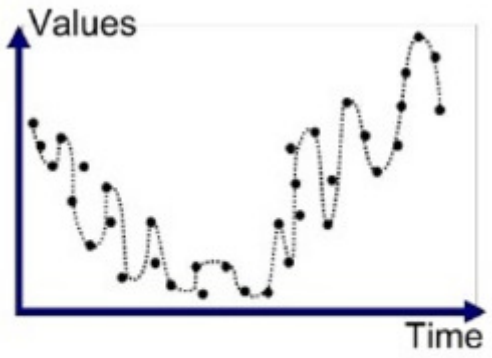
El agente, en su etapa de exploración, realiza acciones que lo llevarán a un estado u otro, donde podría recibir una recompensa o una penalización.

Un ejemplo sería un sistema de recomendación de una página de ventas el cual sería el ambiente. El agente debe decidir qué artículos son los que llamarían más la atención del usuario. En esta situación, que el usuario clickee un artículo recomendado por el agente es la recompensa, siendo una penalización que los artículos sean ignorados. Por medio de ensayo y

error, el agente llegará a un modelo que se adapte a las necesidades de ese usuario en particular sin necesidad de tener información de antemano de dicho usuario.

2.2.4 Overfitting y Underfitting

Ocurre un sobreajuste, también llamado overfitting, cuando el algoritmo de aprendizaje pierde la capacidad de generalizar los patrones en la información y como tal, no aprende ningún patrón extrapolable a nueva información. La causa de este estado es cuando el modelo se centra demasiado en los detalles del conjunto de datos para dar una predicción ideal para dicho conjunto.



Overfitted

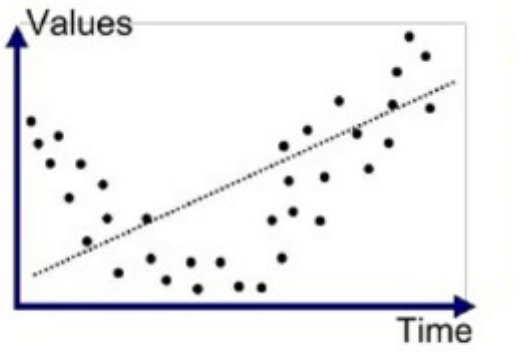
Para su detección, el mejor método es tomar una parte de los datos de entrenamiento y usarlos como un nuevo conjunto de datos en el modelo para validar su desempeño. Si la brecha en los resultados de ambos conjuntos de datos (el de entrenamiento y el nuevo) es muy amplia, es probable que haya un caso de sobreajuste.

Existen varios métodos para evitar esto, los principales son:

- Reducir la complejidad de los datos: una causa común del sobreajuste es el exceso de información.
- Regularización: es una forma de penalizar la complejidad en los modelos, típicamente a nivel de parámetros. Los principales mecanismos de regularización son:
    - L1.
    - L2.
    - Dropout.

- Reducir la fase de entrenamiento: puede ocurrir que un excesivo entrenamiento del modelo lleve al sobreajuste.

Por otro lado, el underfitting (o subajuste) es cuando el modelo es demasiado simple, incapaz de detectar los patrones que lleven a un eficiente aprendizaje.



Underfitted

Se detecta de la misma manera que el anterior, analizando la brecha de la eficacia entre la fase de entrenamiento y la de test. Las principales causas son los modelos de datos demasiado simples, conjunto muy chico de datos y/o un exceso de regularización. Por lo tanto, el camino para evitar el underfitting es recolectando más información o utilizar validación cruzada para aprovechar mejor los datos disponibles.

2.3 Modelos

Se pueden dividir los modelos de aprendizaje supervisados, los usados en el presente proyecto, en dos categorías: regresión y clasificación. La diferenciación entre ambas categorías es su objetivo. Los de regresión buscan una predicción, mientras que los de clasificación buscan categorizar los datos con los que se trabaja. Sin embargo, hay algoritmos híbridos que pueden ser implementados en ambas categorías.

2.3.1 Modelos de regresión

Son modelos matemáticos que buscan determinar la relación entre una variable dependiente, con respecto a otra u otras variables, llamadas explicativas o independientes. En

estos el resultado es un valor numérico, dentro de un conjunto infinito de posibles resultados. Entre las técnicas de Machine Learning que se utilizan en problemas de regresión se destacan: regresión lineal y regresión no lineal, máquinas de vectores de soporte, árboles de decisión, bosques aleatorios y redes neuronales.

Los modelos de regresión (tanto lineal como no lineal) se utilizan en gran medida para predecir valores numéricos como por ejemplo el sueldo. Si la variable independiente es el tiempo entonces se pueden hacer predicciones de valores futuros, sin embargo el modelo puede predecir también valores desconocidos del presente. Las técnicas de Regresión son muy variadas, desde la Regresión Lineal hasta la Regresión de Soporte Vectorial o la Regresión con Bosques Aleatorios.

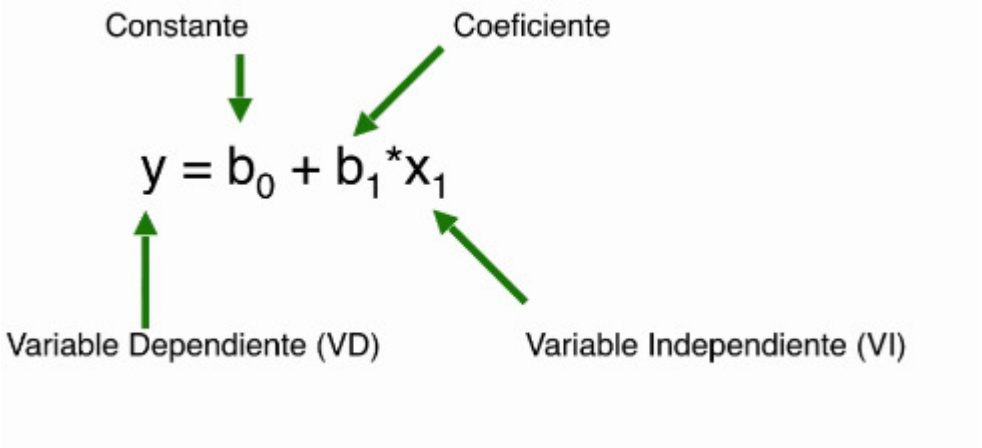
Para entender el por qué de tantos modelos de regresión, se debe hablar de los tipos de variables. Estas pueden ser categorizadas de la siguiente manera:

- Categóricas
    - Nominales: Etiquetas. Ej: Hombre, mujer, rojo, azul, etc.
    - Ordinales: Existe un orden en ella. Ej: Pequeño, mediano, A, B, C, etc.
- Numéricas
    - Discretas: Números enteros. Ej: 1,2,3.
    - Contínuas: Pueden tener decimales. Ej: Edad, altura.

Regresión Lineal Simple

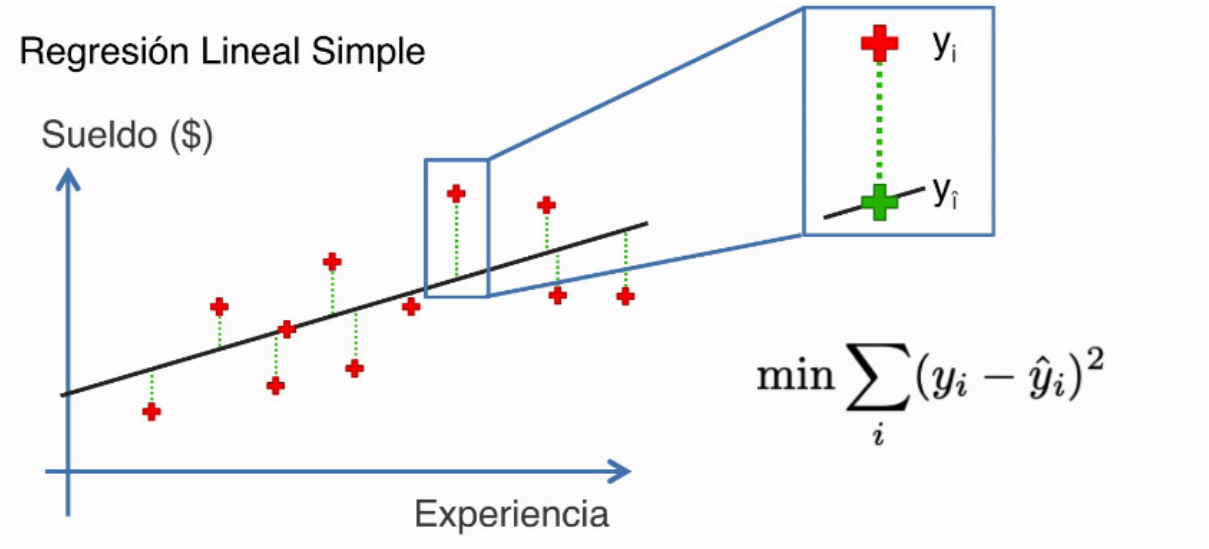
En estadística, se llama análisis de la regresión al proceso estadístico de estimar las relaciones que existen entre las variables. Su eje es el de estudiar la relación entre una variable dependiente con una o más variables independientes.

En el modelo de regresión lineal simple, sólo existe una variable independiente la cual se visualiza en el siguiente ejemplo:



- Variable dependiente: el valor que se busca predecir con el algoritmo.
- Variable independiente: un valor que no está restringido por el resto de la ecuación.
- Constante: la ordenada en el origen. Es el valor inicial cuando la variable independiente es 0.
- Coeficiente: indica la pendiente de la recta.

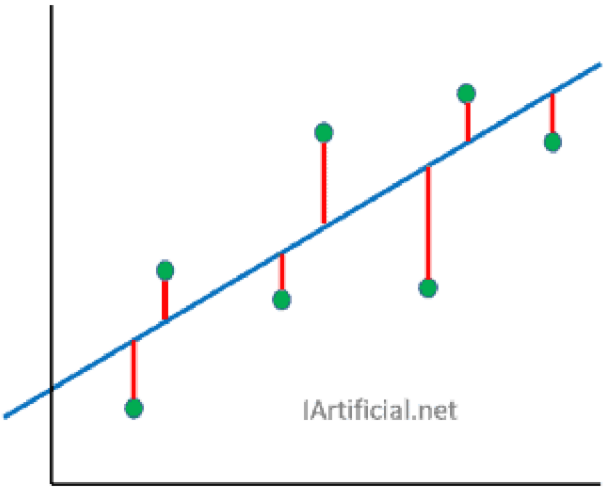
En este modelo, se recrea una ecuación como la mencionada como ejemplo que da como resultado una recta. El algoritmo tiene como objetivo encontrar la recta más cercana a los datos que recibe de entrada. Dicha recta representa la predicción que retornará el modelo.



En el ejemplo anterior, los puntos rojos son los resultados reales de la predicción. Al ser los puntos dispares y el modelo limitado a una recta, es muy poco probable que logre

acertar un 100% a todos los puntos por lo que habrá un margen de error (la distancia entre Yi e Yi).

Para saber cuál ecuación es la más óptima, se calcula el error cuadrático medio. Para entender el concepto se puede ver el siguiente ejemplo.



En la figura se ve la predicción de la regresión lineal (en azul) para los datos trabajados (los puntos verdes). El error entre la predicción y el valor real se representa en rojo que es la distancia entre ambos valores.

$$error\ cuadrático = (real\ -\ estimado)^2$$

Se calcula el error al cuadrado para que siempre sea positivo. De ésta forma, se sabe que el error perfecto es 0. Para obtener el error cuadrático medio, se suman todos los errores cuadráticos y se divide entre el número total de puntos(en la ecuación representada con M) quedando de la siguiente manera:

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(real_i\ -\ estimado_i)^2$$

El modelo usa este resultado para comprobar la efectividad de la predicción. Cuanto menor sea el valor, mayor será la tasa de acierto de la predicción lineal.

Regresión Lineal Múltiple

Comparte la lógica con el modelo de regresión lineal simple la cual es realizar una predicción lineal con el mínimo margen de error posible. La principal diferencia, como se indica en el nombre, es que en esta ocasión se cuenta con dos o más variables independientes, también conocidas como predictores.

Cuando se trata con una regresión lineal múltiple, existen tres problemáticas principales que son:

- La predicción se ve perjudicada por la incorporación de predictores correlacionados.
- Inclusión indiscriminada de predictores, rellenando el modelo con valores que aporten poca relevancia a la predicción, perjudicando el rendimiento de éste.
- No pueden ajustarse cuando el número de predictores es superior al número de observaciones.

Algunas de las estrategias que se pueden aplicar para evitar el impacto de las problemáticas mencionadas son:

- Subset selection: utilizar un proceso iterativo que vaya descartando los predictores menos relevantes. Existen varios métodos para este proceso, siendo los principales Best subset selection y stepwise selection. En un esquema general, el proceso consiste en tres partes:
    a. Crear un conjunto de modelos candidatos mediante distintas combinaciones de los predictores disponibles.
    b. Para cada posible tamaño de modelo, se selecciona el mejor basándose en el error de entrenamiento.
    c. Los modelos finalistas de cada tamaño se comparan entre sí mediante una métrica de validación y se identifica el mejor.
- Regularización: método que fuerza a los coeficientes del modelo a tender a cero, minimizando el riesgo de overfitting (efecto de sobreajustar al modelo con los datos conocidos y que sólo sea efectivo con estos, reduciendo su rendimiento ante nueva información). La regularización también reduce la influencia de los predictores menos relevantes en la predicción final. Los principales son: la regularización de Ridge y Lasso. La principal diferencia práctica entre ambos es que el primero consigue que algunos coeficientes sean exactamente cero, por lo que realiza selección de

predictores; mientras que el segundo no llega a excluir ninguno. Esto supone una ventaja notable de Lasso en escenarios donde no todos los predictores son importantes para el modelo y se desea que los menos influyentes queden excluidos.

Por otro lado, cuando existen predictores altamente correlacionados (linealmente), Ridge reduce la influencia de todos ellos a la vez y de forma proporcional, mientras que Lasso tiende a seleccionar uno de ellos, dándole todo el peso y excluyendo al resto. En presencia de correlaciones, esta selección varía mucho con pequeñas perturbaciones (cambios en los datos de entrenamiento), por lo que, las soluciones de lasso, son muy inestables si los predictores están altamente correlacionados.

- Reducción de la dimensionalidad: reduce el número de predictores al crear nuevos a partir de combinaciones de los predictores originales.

Para que un modelo de regresión lineal múltiple sea efectivo, debe cumplir las siguientes condiciones:

- Linealidad.
- Homocedasticidad.
- Normalidad multivariable.
- Independencia de los errores.
- Ausencia de multicolinealidad.

2.3.2 Modelos de clasificación

Se diferencian de los modelos de regresión, donde se busca predecir un valor continuo, al utilizarse como objetivo para la predicción una serie finita de categorías (ejemplo: 1 y 0, sí y no, bajo-mediano-alto). Existen una gran amalgama de aplicaciones del proceso de clasificación desde medicina hasta marketing. Como ejemplo cotidiano de ésto, existen las recomendaciones por parte de las plataformas de streaming que, por medio de un algoritmo en base a una experiencia previa, califican qué películas o series pueden ser del interés del usuario. Para ello, el resultado de una predicción retorna entre una serie de valores finitos que serían el total de categorías disponibles.

K-NN

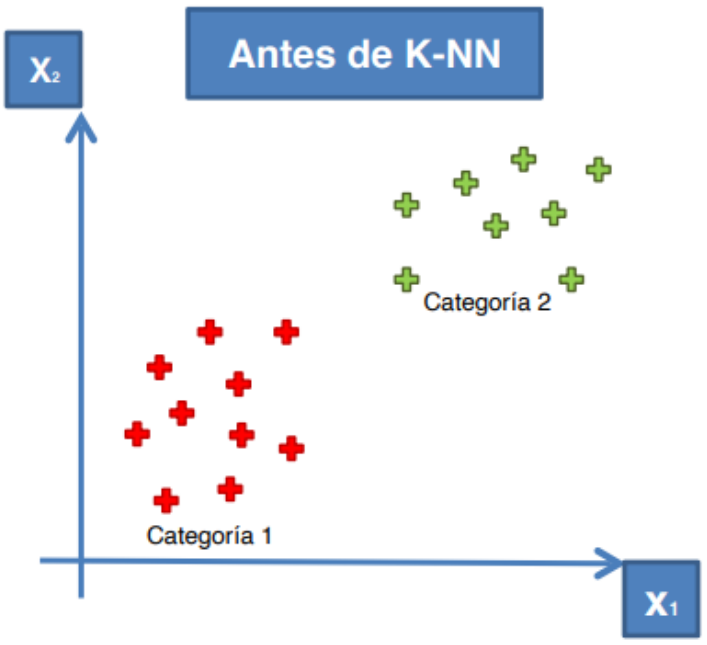
El método k vecinos más cercanos o K-NN (del inglés k-nearest neighbors) es un método de clasificación supervisada (aprendizaje basado en un conjunto de entrenamiento). Estima la probabilidad de que un elemento x pertenezca a una categoría a partir de la información ya proporcionada por el conjunto de entrenamiento.

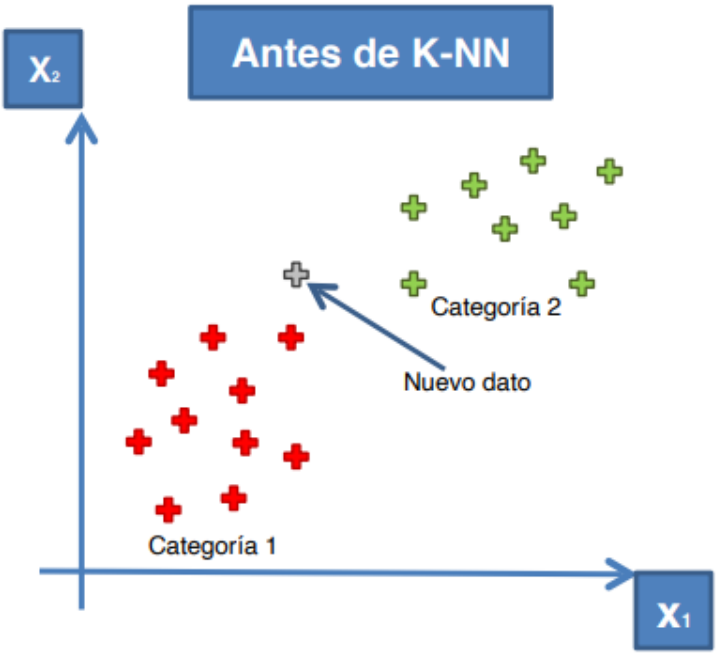
El criterio de selección de categoría parte, como indica su nombre, de la cantidad k de vecinos cercanos alrededor del nuevo dato. Si la mayoría de estos valores pertenecen a una categoría, entonces la nueva información se agrupará a dicha categoría.

Cabe señalar que k es un número arbitrario el cual se recomienda que sea impar para evitar empates.

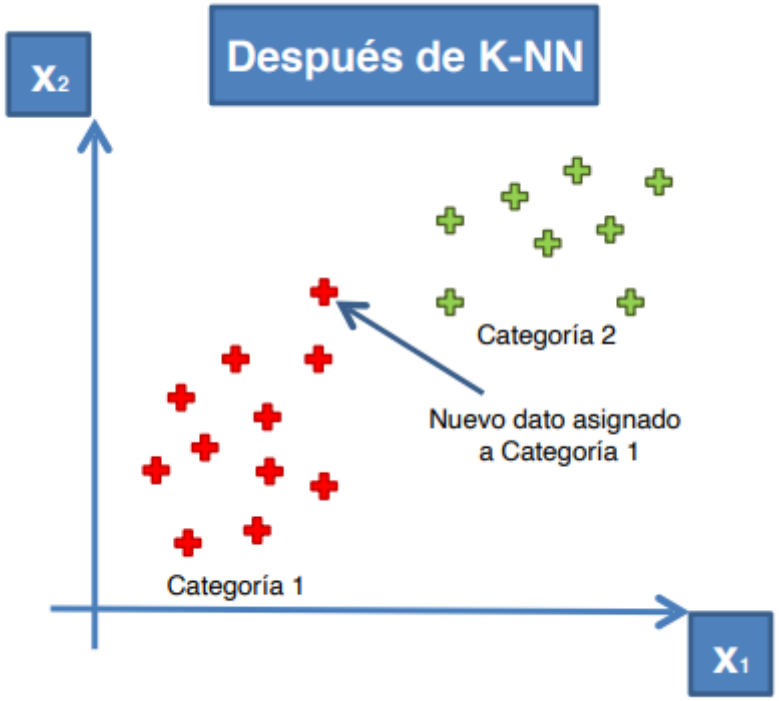
Funcionamiento:

1. Se elige un valor K de vecinos a tener en cuenta. Para evitar los empates, se recomienda un número impar.
2. Se toma los vecinos K más cercanos del nuevo dato según la distancia Euclídea. En este ejemplo, K=5.





3. Entre esos K vecinos cercanos, se cuenta el número de puntos que pertenencen a cada grupo de datos (divididos en su fase de entrenamiento por la categoría a la que pertenecen). En el ejemplo tenemos tres vecinos de la Categoría 1 y dos de la Categoría 2.
4. Se asigna el nuevo dato a la categoría con más vecinos.



Y así el modelo ya está completo.

2.3.3 Modelos híbridos

Son modelos que pueden funcionar tanto en ejercicios de clasificación como de regresión. En el presente trabajo los modelos híbridos utilizados son: Árboles de decisión, Bosques aleatorios y Redes Neuronales Artificiales.

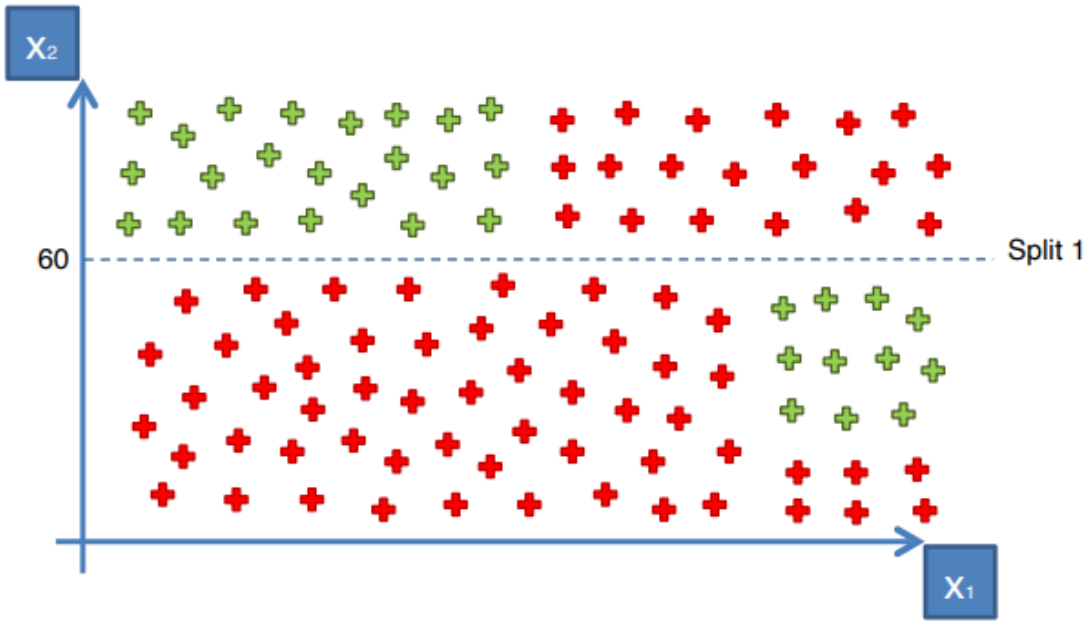
Árboles de decisión

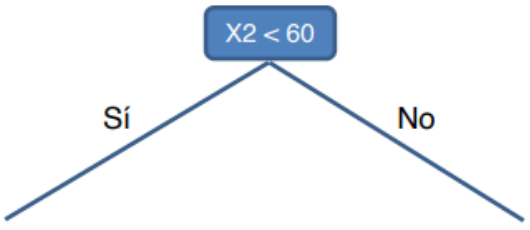
CART (A Classification And Regression Tree) es una técnica de clasificación y predicción basándose en la agrupación y categorización de datos. Son algoritmos para clasificar utilizando particiones sucesivas de los datos; apropiados para cuando hay un número elevado de información, una de sus ventajas es su naturaleza descriptiva que permite interpretar las decisiones tomadas por el algoritmo.

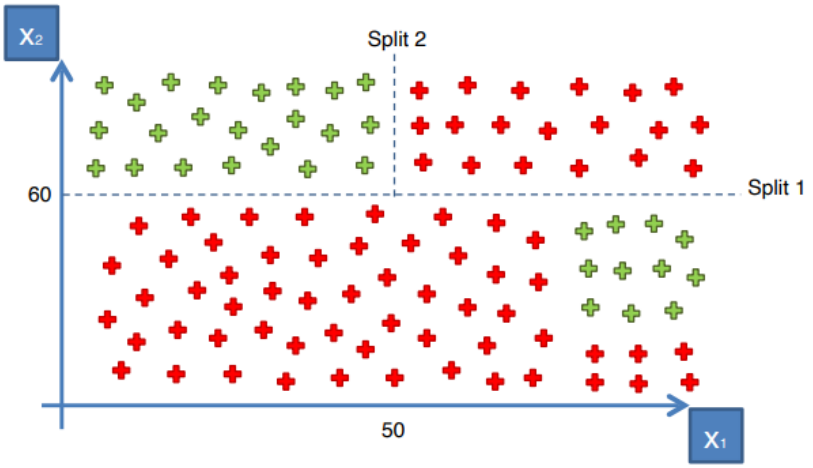
Su funcionamiento consiste en crear diagramas de construcciones lógicas para resolver las problemáticas ofrecidas al modelo. Dicha técnica se le denomina segmentación jerárquica que utiliza un proceso de división secuencial, iterativo y descendente que,

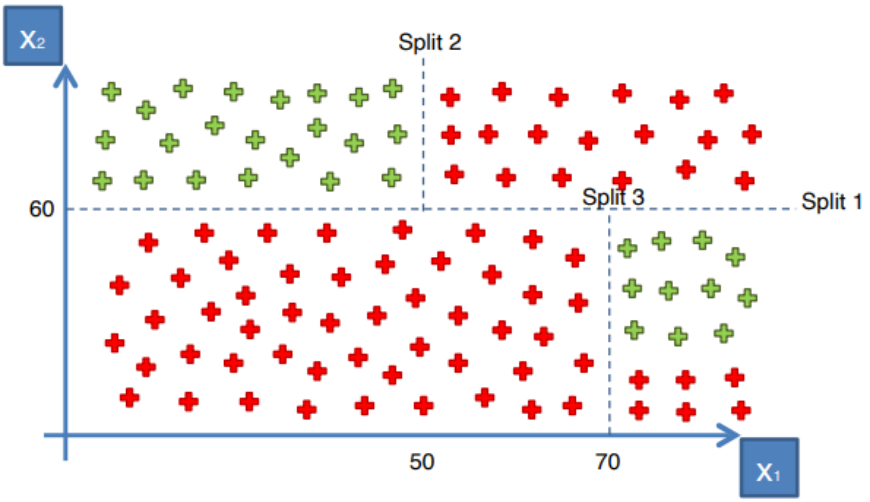
partiendo de una variable dependiente, forma grupos homogéneos definidos específicamente mediante combinaciones de variables independientes en las que se incluye la totalidad de los recogidos en la muestra.

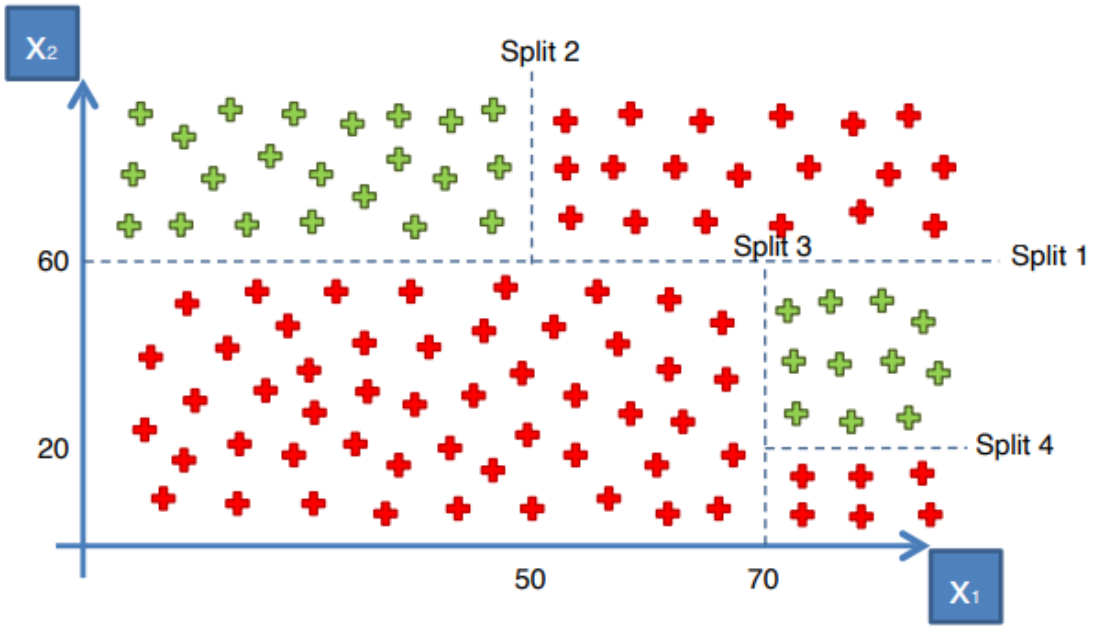
El criterio que utiliza para dividir los segmentos consiste en maximizar el número de elementos de una categoría dentro de cada división. En el siguiente ejemplo, busca dividir los puntos verdes de los rojo, siendo cada color una categoría y, a partir de entonces, irá formando un árbol de decisión:

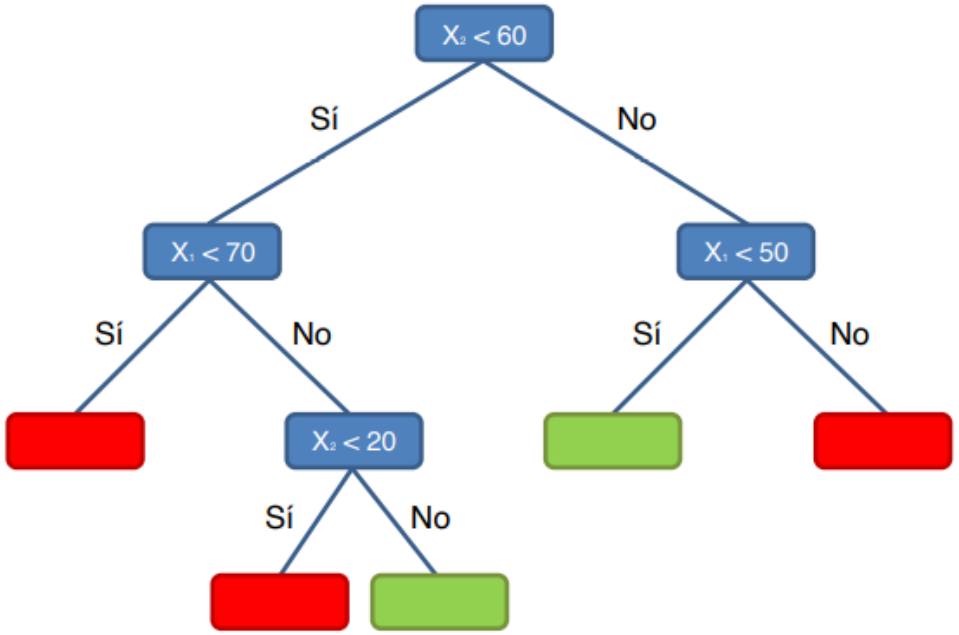





X2
Split 2
60
Split 1
50
X1


X2
Split 2
60
Split 3
Split 1
50
70
X1





Dependiendo de la cantidad de divisiones, los árboles pueden llegar a ser muy costosos, por lo que se les consideraba un método antiguo. Sin embargo, hay algoritmos que los han reconvertido y mejorado con los años con técnicas como Random Forest o Gradient Boosting.

Por último, una problemática común de los árboles es el sobreajuste. Ésta ocurre cuando entre los datos surgen incoherencias o casos excepcionales. El algoritmo, en vez de ignorarlas, buscará ajustarse a estas predicciones pudiendo retornar datos erróneos. Para solucionar este problema hay que limitar el crecimiento del árbol para conseguir modelos más generales.

Bosques aleatorios

Cuando se dice Bosques aleatorios, se habla de aprendizaje en conjunto: una serie de algoritmos que, combinando distintos modelos, se obtiene un mejor resultado que la suma de sus partes. En el caso de los Bosques aleatorios, es una combinación de N árboles de decisión donde cada uno hará una predicción y el sistema retornará la categoría con más votos.

Proceso:

1. Seleccionar un número aleatorio K de puntos del Conjunto de Entrenamiento.
2. Construir el Árbol de Decisión asociado s dicho conjunto K.
3. Elegir un número N de árboles que queremos construir y repetimos los pasos 1 y 2 hasta crear la cantidad N.
4. Para clasificar un nuevo punto, cada N árbol elabora una predicción a la categoría que pertenece, retornando el valor con más votos.

Los bosques aleatorios no tienen el problema del sobreajuste que sí tienen los árboles de decisión. Ésto se debe a que, al tomar el resultado con más votos, suele ignorar las excepciones que pueden retornar un árbol o dos.

Los principales hiperparámetros en éste modelo son:

- Estimadores: cantidad n de árboles que va a tener el bosque aleatorio. Normalmente cuantos más mejor, pero a partir de cierto punto deja de mejorar y sólo hace que vaya más lento. Un buen valor por defecto puede ser el uso de 100 árboles.
- Jobs: número de cores que se pueden usar para entrenar los árboles. Cada árbol es independiente del resto, así que entrenar un bosque aleatorio es una tarea muy paralelizable. Por defecto sólo utiliza 1 core de la CPU. Para mejorar el rendimiento se puede usar tantos cores como see estimen necesario.

- Características máximas: se refiere al número máximo de características que los Bosques Aleatorios considera para dividir un nodo. Una forma de garantizar que los árboles son diferentes, es que todos se entrenan con una muestra aleatoria de los datos. Si se desea una mayor diferenciación, se puede hacer que distintos árboles usen distintos atributos. Esto puede ser útil especialmente cuando algunos atributos están relacionados entre sí.

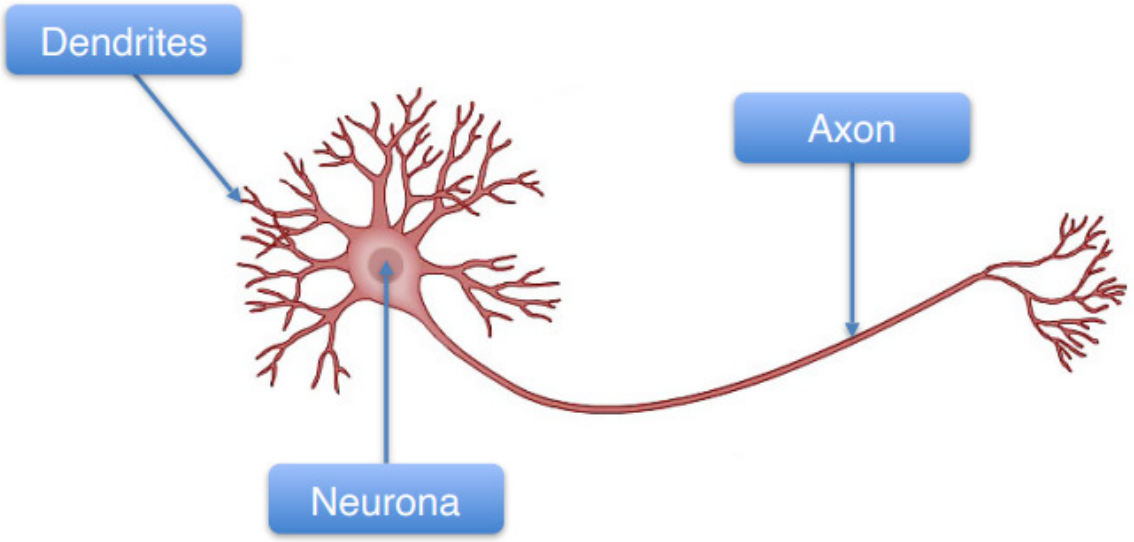
En cuanto a la regularización de los bosques, las siguientes propiedades definen su arquitectura:

- Profundidad: La máxima del árbol.
- Mínimo de muestras divisorias: El número menor posible de muestras necesarias antes de dividir este nodo. También se puede expresar en porcentaje.
- Mínimo de muestras por nodo: La menor cantidad que debe haber en un nodo final (hoja). También se puede expresar en porcentaje.
- Máximo de nodos: El mayor número posible de nodos finales.

2.3.4 Redes Neuronales Artificiales

Es un modelo matemático inspirado en el comportamiento biológico de las neuronas y en cómo se organizan formando la estructura del cerebro.

Para entender el funcionamiento de una neurona, se puede, de manera simplificada, describir su estructura dividiéndola en tres partes:



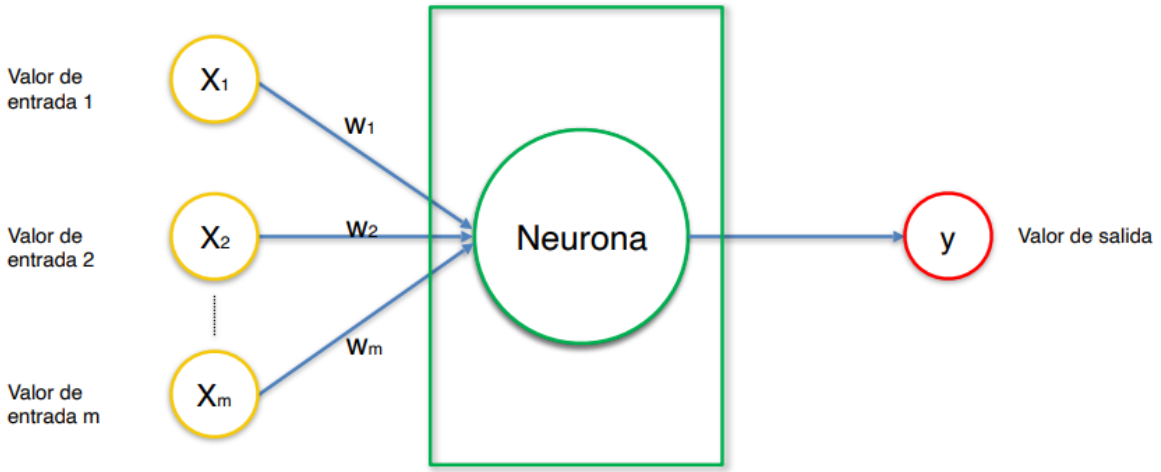
- Dendrites: Receptores de entradas.
- Neurona: Cuerpo principal y encargado de procesar los datos de entrada.
- Axon: Conector de la neurona. Transmite la señal en forma de impulsos eléctricos y químicos.

De la misma manera, se compone de nodos que pueden o recibir información, procesarla, o enviarla a otro. Combinando estos nodos, también llamados neuronas, se crea una red que se va retroalimentando para llegar a una organización que retorne la mejor predicción.

Cada nodo cuenta con al menos un receptor de entrada (también llamado input) de donde recibe la información proveniente de otra neurona.

Al mismo tiempo, cada uno se conecta entre sí mediante un conjunto de trayectorias ponderadas. Esto significa que cada conexión vendrá asociada a un peso, que representa la significancia del input proveniente de dicha conexión. Cuanto mayor peso, mayor significancia y, por lo tanto mayor, impacto tendrá el input en el valor de salida.

Para ajustarse, el modelo evalúa los errores de predicción y modifica los pesos de cada conexión en el proceso de testeo. Luego, vuelve a realizar el proceso y los analiza nuevamente. El ciclo se repite varias veces hasta haberse equilibrado en un estado donde las predicciones son óptimas.



Con el tiempo, el nodo irá ajustando los pesos de cada entrada, marcando cuál es la más relevante para su función. Es el ajuste de estos pesos el principal proceso de aprendizaje de la red y su criterio se basa en las funciones de activación que contiene cada nodo, con el que mide los resultados y los pesos de cada entrada.

La neurona recibe n datos de entrada (x1, x2, x3... xn) y calcula el peso ponderado de estos valores. Los pesos a aplicar son w1, w2, w3... wn, cifra a la que se suma un valor adicional w0 que recibe el nombre de bias. Todos los pesos y el valor de w0 son "aprendidos", es decir, se escogen durante el entrenamiento de la neurona. El resultado de la aplicación de esta función lineal es:

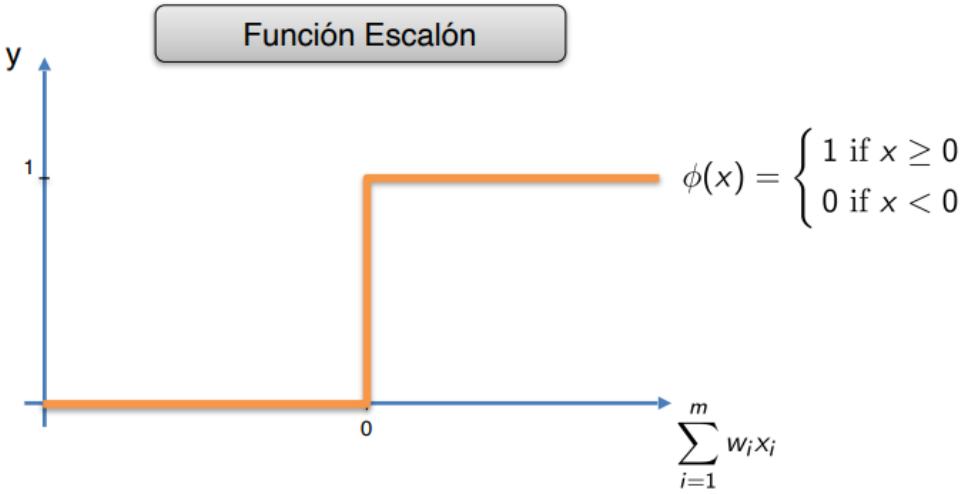
a = w0 + x1w1 + x2w2 + x3w3 + ... + xnwn

Dicho resultado será procesado por la función de activación para obtener un valor de salida acorde al aprendizaje.

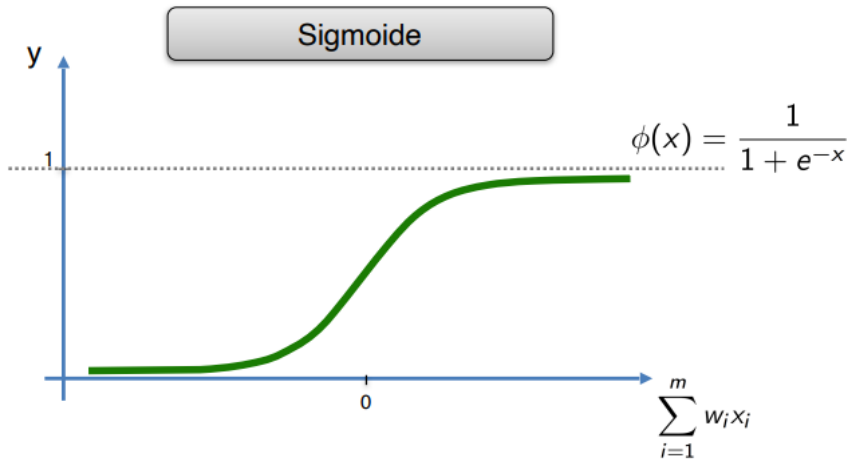
Función de activación

Es una función continua que se le aplica el resultado al cálculo de los pesos y las entradas para evitar la linealidad. Esto provoca una transformación de los datos dependiendo de la función se escoja. Existen varios tipos de función y es elegida acorde a la tarea asignada a la neurona. Las más comunes son:

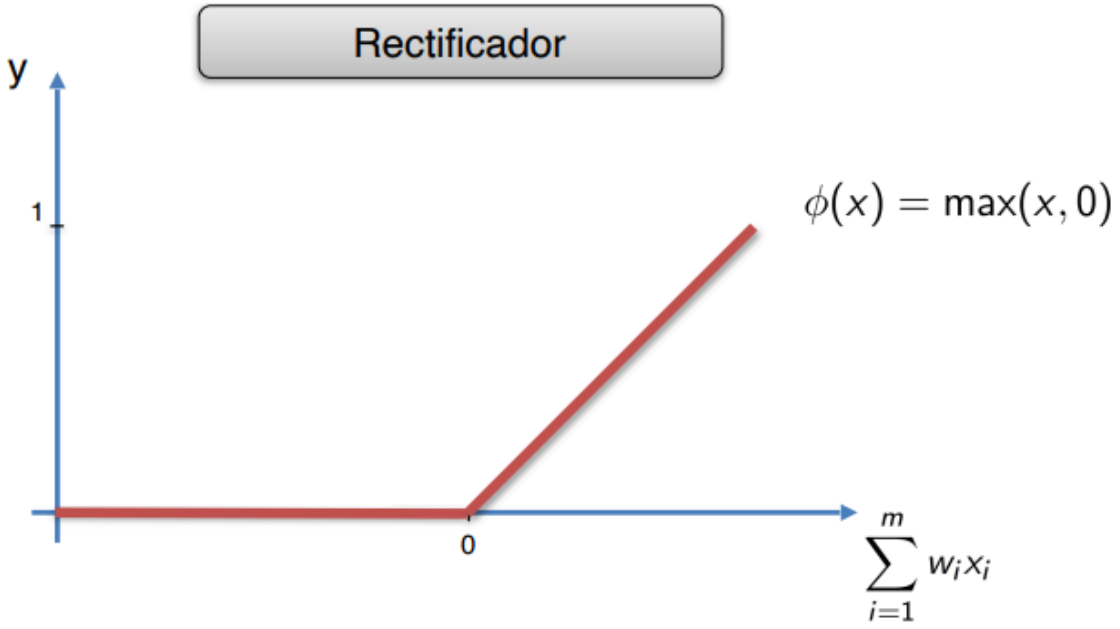
- Función escalón: devuelve 1 o 0 si la ponderación da positiva o negativa, respectivamente. Se usa cuando se quiere clasificar o se tiene salidas categóricas.



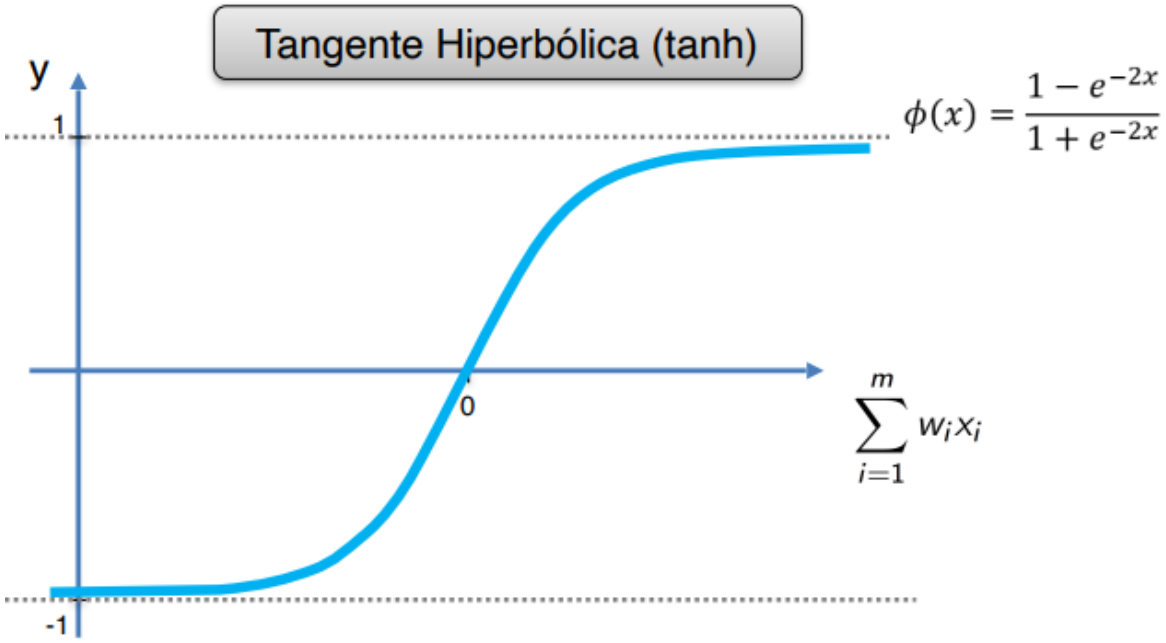
- Sigmoide: También llamada función logística, su valor tiende entre cero y uno por lo que la salida es interpretada como una probabilidad. En el ejemplo, si los valores son muy negativos, es decir x<0, la función será igual a cero; si se evalúa en cero la función dará 0.5 y en valores altos su valor es aproximadamente 1. Suele usarse en la última capa y se usa para clasificar datos en dos categorías.



- Rectificador: max(x,0): Transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran. Es la función más utilizada en los proyectos de redes neuronales artificiales pues permite un rápido aprendizaje en las redes neuronales. Debido a la anulación de los valores menores a cero, puede provocar la muerte de las neuronas (que estén inactivas al no retornar nada). Por esta razón la función ReLu tiene una variante denominada Leaky ReLu que va a prevenir que existan neuronas muertas debido a la pequeña pendiente que existe cuando x<0.



- Tangente hiperbólica: similar a la función sigmoide pero va de -1 a 1. Utilizada para clasificar datos. La red nos dará la probabilidad de que el valor de entrada pertenezca a cada categoría existente, sumando en su total 1. El resultado que retornará es la categoría con mayor probabilidad. Usado en casos de categorías múltiples.



Cada neurona puede tener una función distinta de la otra, pudiendo tener una red neuronal trabajando con múltiples funciones de activación.

Contienen además, capas ocultas que se encargan de aumentar la precisión, funcionando como intermediarios entre los valores de entrada con los de salida. Cada nodo de la capa oculta puede tomar distintos valores de entrada para el análisis.

Aprendizaje

El juicio que aplica la RNA para decidir qué peso debe modificarse en la siguiente corrección radica en la función de coste. Al igual que la función de activación, existen varios métodos para definir dicha función siendo la más común:

$$C = \frac{1}{2}(\hat{y} - y)^2$$

Siendo y el valor real e ŷ la predicción. Cuanto mayor el coste, peor es la predicción. El objetivo del entrenamiento es medir el coste para luego ir iterando hasta que el algoritmo descubra los parámetros del modelo con la pérdida más baja posible. Una vez que esto ocurra el modelo ha convergido.

Un método para reducir el coste es el descenso de gradientes, el cual es un algoritmo que permite conocer los parámetros correctos para tener el coste mínimo. Este, a la vez, varía en los siguientes tipos:

- Descenso del gradiente en lotes (o batch): todos los datos disponibles se introducen de una vez. Esto supondrá problemas de estancamiento ya que el gradiente se calculará usando siempre todas las muestras y llegará un momento en que las variaciones serán mínimas.
- Descenso del gradiente estocástico: se introduce una única muestra aleatoria en cada iteración. El gradiente se calculará para esa muestra concreta, lo que supone la introducción de la aleatoriedad, dificultando así el estancamiento. El problema de esta versión es su lentitud, ya que necesita de muchas más iteraciones.
- Descenso del gradiente (estocástico) en mini-lotes (o mini-batch): en lugar de alimentar la red con una única muestra, se introducen N muestras en cada iteración; conservando las ventajas de la segunda versión y consiguiendo además que el entrenamiento sea más rápido debido a la paralelización de las operaciones.

Propagación hacia atrás

Es la fase final para ajustar los pesos. en base a los errores de cada ciclo.

Proceso:

1. Inicializar los pesos aleatoriamente con valores cercanos a 0 (pero no 0).
2. Introducir la primera observación del conjunto de datos en la Capa de Entrada, cada característica es un nodo de entrada.
3. Propagación hacia delante: de izquierda a derecha, las neuronas se activan de modo que la activación de cada una se limita por los pesos. Propaga las activaciones hasta obtener la predicción y.
4. Comparamos la predicción con el resultado final. Se mide entonces el error generado.
5. Propagación hacia atrás: de derecha a izquierda, propagando el error hacia atrás. Se actualizan los pesos según lo responsables que sean del error. El ratio de aprendizaje gobierna cuánto deben actualizarse los pesos.
6. Se repiten los Pasos 1 a 5 y se actualizan los pesos después de cada observación (Reinforcement Learning). O se repiten los pasos 1 a 5 pero se actualizan los pesos después de un conjunto de observaciones (Batch Learning).
7. Cuando todo el conjunto de Entrenamiento ha pasado por la RNA, se completa este ciclo llamado epoch.
8. Repetir epochs para aumentar el ajuste hasta donde se crea necesario.

3 Herramientas de trabajo

3.1 Especificaciones

Los ejercicios fueron realizados en un equipo con las siguientes características:

Sistema operativo	Windows 10 Pro 64-bit
Procesador	AMD Ryzen 7 3700X 8-Core Processor 3600 Mhz
Memoria RAM	32,0 GB

Versión de DirectX	DirectX 12
Tarjeta gráfica	AMD Radeon RX 5600 XT

3.2 Python

Es un lenguaje de alto nivel de programación interpretado. Puede ser utilizado tanto para crear servidores, aplicaciones de escritorio y scripts de comandos, entre otros. Es multiplataforma por lo que puede ser instalado en cualquier Sistema Operativo desde el siguiente link: https://www.python.org/downloads/

Además de su versatilidad, comunidad e hincapié en la legibilidad de su código, Python cuenta con un extenso conjunto de librerías especializadas para trabajar con Machine Learning. Las utilizadas en este trabajo son:

- numpy: da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.
- pandas: extensión de numpy para manipulación y análisis de datos. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.
- sklearn: librería que permite gestionar el preprocesado de datos, la creación de modelos supervisados y no supervisados, y el entrenamiento de estos.

3.3 Software

Anaconda

Es una distribución libre y abierta de los lenguajes Python y R, ambos utilizados en ciencia de datos, y aprendizaje automático. Está orientado a simplificar el despliegue y administración de paquetes.

Viene integrado por defecto con los paquetes más populares de Python así como con herramientas como Ipython, Jupyter Notebook y Spyder IDE, entre otros. En el proyecto se utilizó la herramienta Spyder IDE.

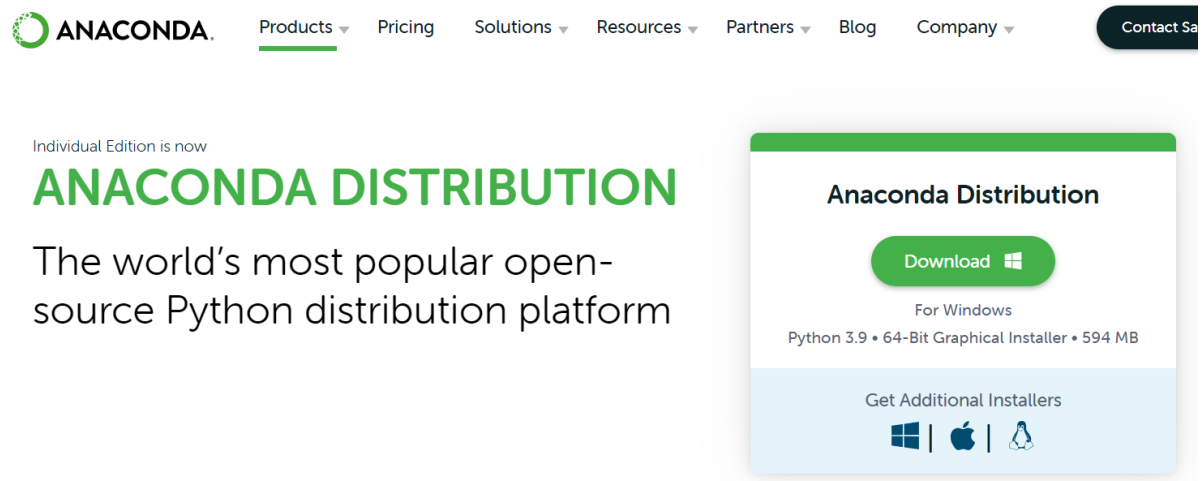
Spyder es un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) que permiten escribir scripts de Python e interactuar con el software de Python desde una interfaz única.

Spyder IDE

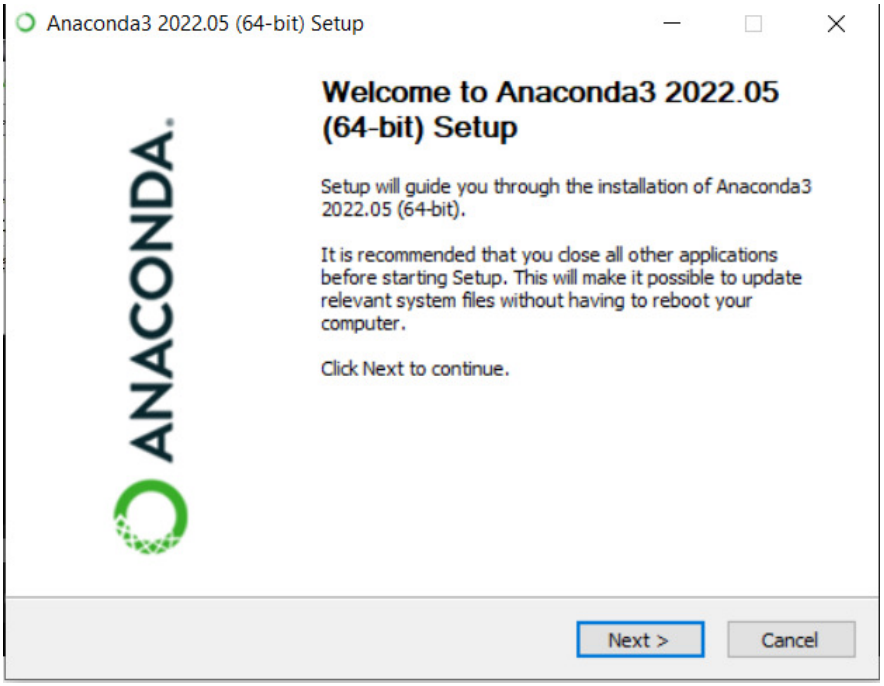
Spyder es un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) que permite escribir scripts e interactuar con el software de Python desde una interfaz única. Escrito en Python y diseñado por y para científicos, ingenieros y analistas de datos, ofrece una combinación de características que permite el análisis y ejecución de la inspección de datos

3.4 Implementación de entorno de trabajo

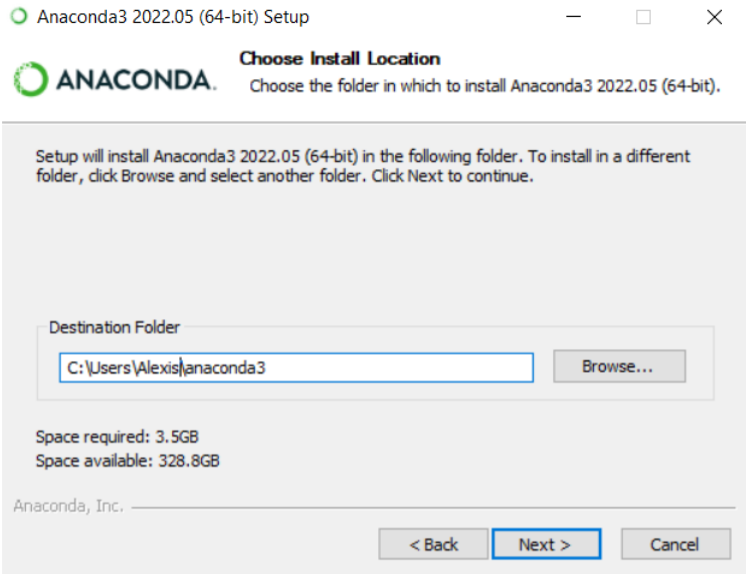
- Para instalar Anaconda, primero se debe descargar la distribución desde el siguiente link: https://www.anaconda.com/products/distribution

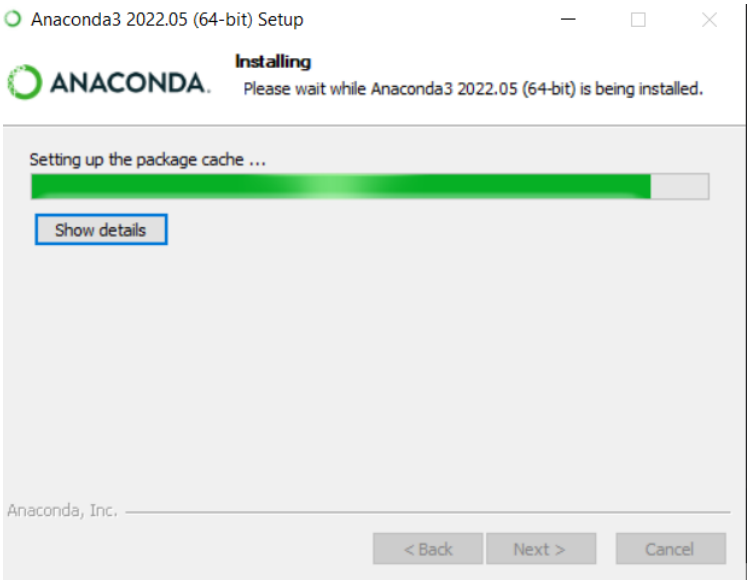


- Descargado y ejecutado la distribución, se da comienzo al proceso de instalación:

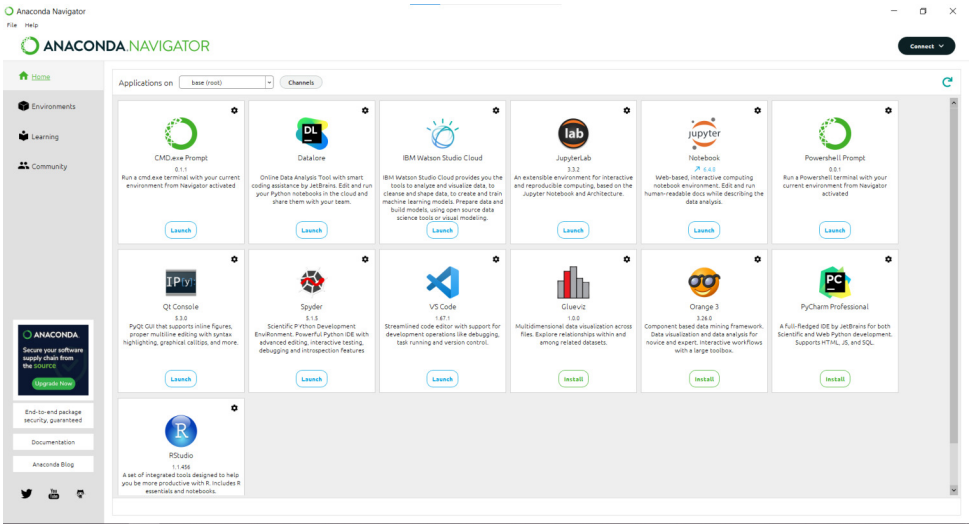


- Se indica la ubicación donde se quiere instalar y se da Continuar:

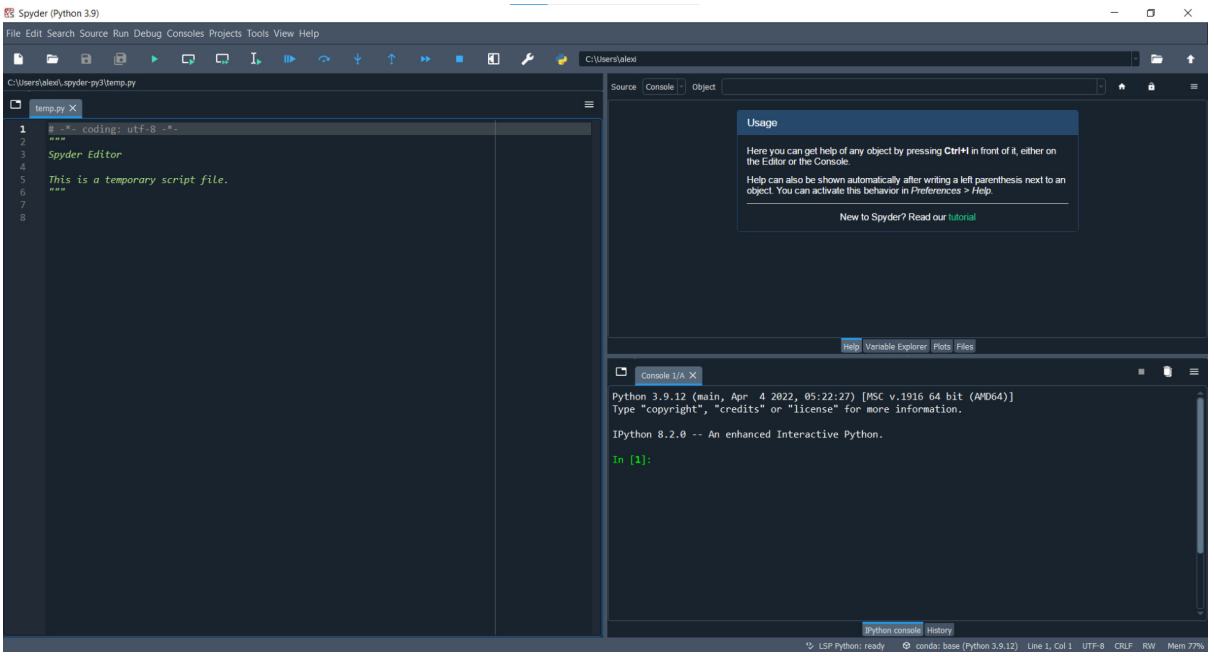




- Terminado el proceso, el software se guardará como "Anaconda Navigator" y en su interior se hallan las distintas herramientas que contiene.



- Para el proyecto, se trabajó con el IDE Spyder que lee y ejecuta archivos Python (con la extensión .py):



- Para ejecutar el archivo actual, se presiona CTRL + Enter.

4. Preprocesado de datos

Validación cruzada

La validación cruzada o cross-validation es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la predicción y se quiere estimar la precisión de un modelo que se llevará a cabo a la práctica.

Variables Dummy

Es un proceso que traduce y reemplaza las variables categóricas (como ejemplo: sexo, ciudad o país) en nuevos campos numéricos donde cada uno representa un valor de una serie finita. La categoría a la que pertenece cada registro es marcada con un 1 mientras que al resto de los campos con 0.

Como ejemplo, se tiene una tabla de 5 registros con el campo sexo cuyo valor solo puede “female” o “male”.

Index	Sexo
1	female
2	male
3	male
4	female
5	male

Aplicando el proceso de las variables dummy quedaría de la siguiente manera:

Index	Female	Male
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Un factor importante a la hora de crear las variables dummy es que la cantidad de nuevas columnas deben ser la cantidad de categorías n menos uno. Esto se hace para evitar la redundancia en los datos pues, por eliminación y siguiendo el ejemplo, si un registro no pertenece a la categoría “female” por descarte entra en la categoría “male”. De ésta forma se evita la multicolinealidad que es un exceso de dependencia entre dos columnas que puede producir lecturas erróneas de la información y aumentar la probabilidad de fallos en la

predicción. Por lo tanto en el ejemplo, una correcta aplicación del proceso de las variables dummy es el siguiente:

Index	Female
1	1
2	0
3	0
4	1
5	0

Coeficiente de correlación de Pearson

Se conoce como correlación entre dos variables la dependencia que comparten ambas entre sí. Si tienen una correlación positiva, significa que al aumentar un valor, lo mismo ocurrirá con el otro. En caso de ser negativa, al aumentar un valor, disminuirá el otro.

En el presente trabajo se utilizó el coeficiente de correlación de Pearson que sirve para categorizar el grado de correlación entre cada campo y el valor final a predecir.

La ecuación para calcular el coeficiente de correlación de Pearson es el siguiente:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \ \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Descripción de los datos

Regresión

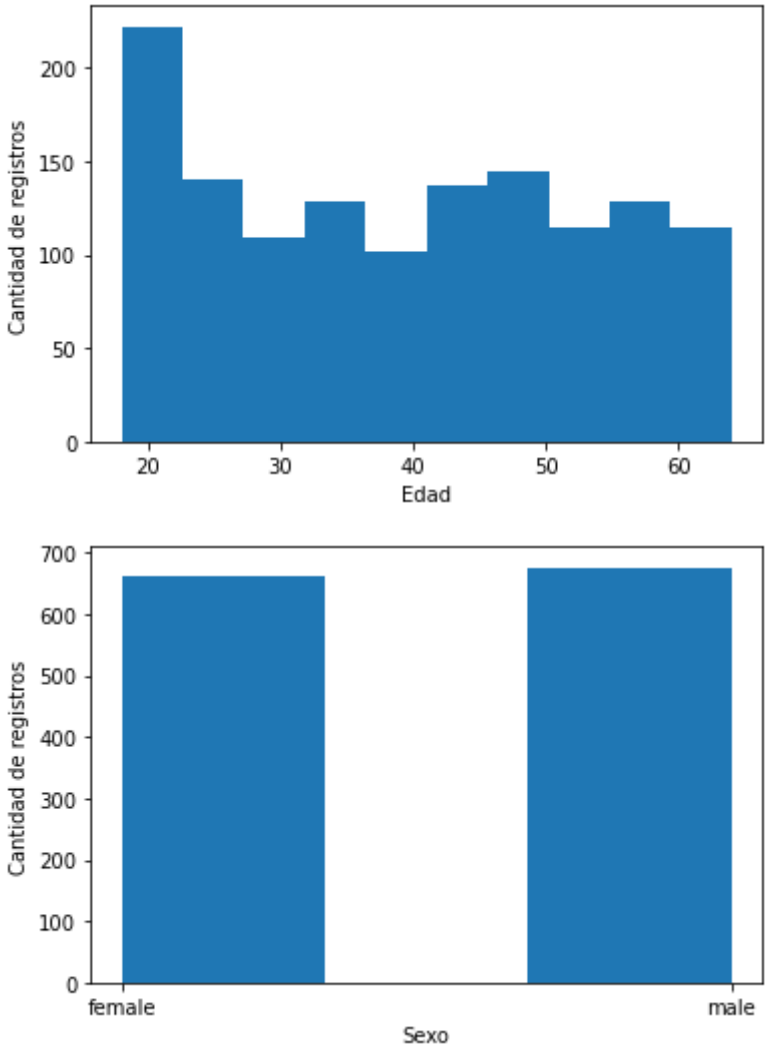
Se tiene un conjunto de datos, también llamado dataset, con 1338 registros y 8 campos.

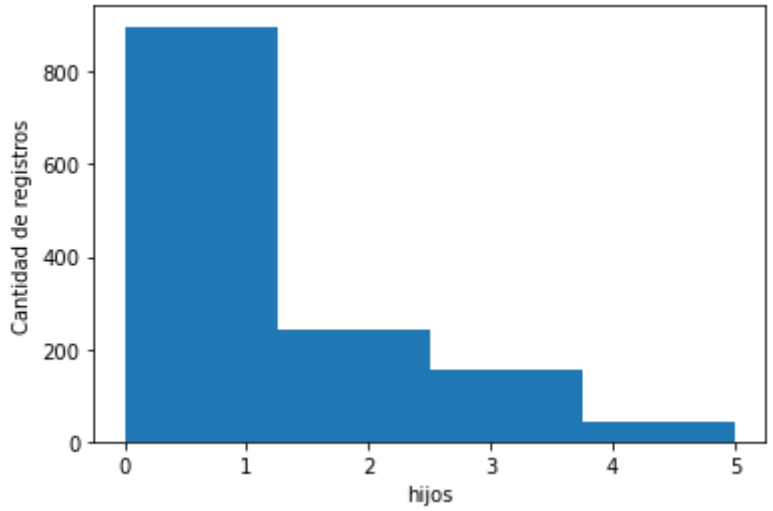
Campo	Tipo de valor
Edad	Numérico
Sexo	String
Edad	Numérico
Hijos	Numérico
Fumador	String
Región	String
Cargos	**Numérico, valor a predecir**

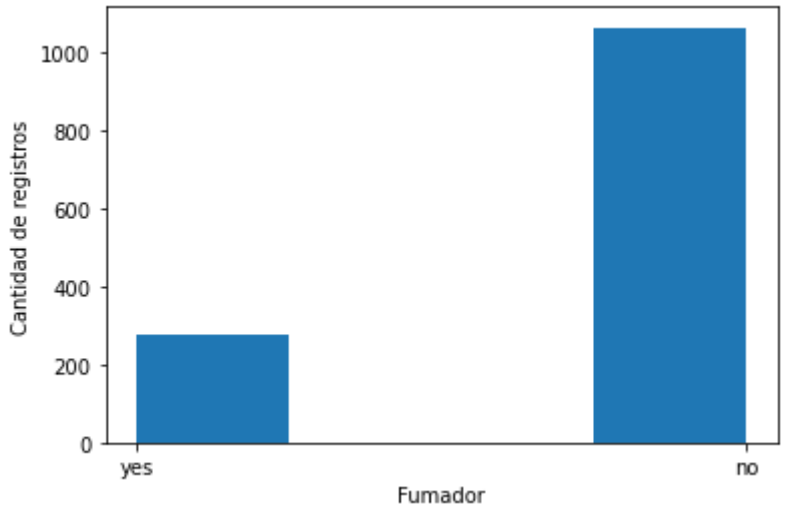
Campo	Total	Media	Desviación standard	Mínimo	25%	50%	75%	Máximo
Edad	1338	39.207	14.05	18	27	39	51	64
Edad	1338	30.6634	6.09819	15.96	26.2963	30.4	34.6938	53.13
Hijos	1338	1.09492	1.20549	0	0	1	2	5

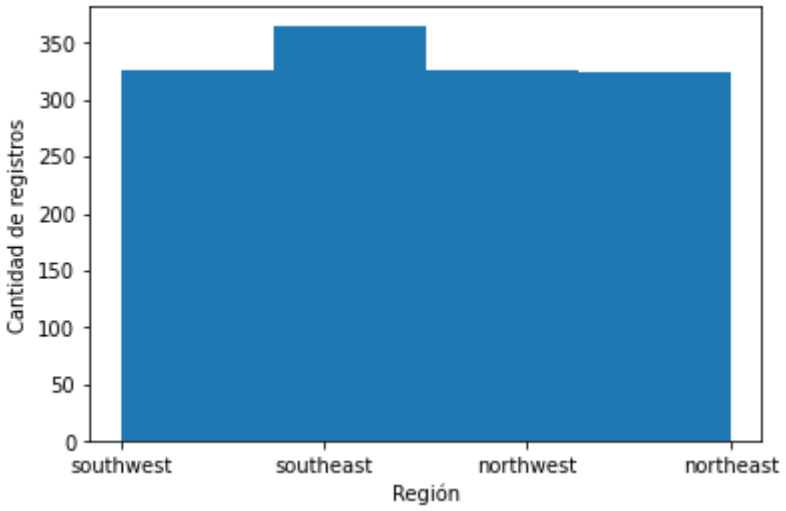
Cargos	1338	13270.4	12110	1121.87	4740.29	9382.03	16639.9	63770.4

Distribución gráfica




Cantidad de registros
800
600
400
200
0
0
1
2
3
4
5
hijos


Cantidad de registros
1000
800
600
400
200
0
yes
no
Fumador


Cantidad de registros
350
300
250
200
150
100
50
0
southwest
southeast
northwest
northeast
Región

Aplicación de las variables dummy

Existen 3 campos categóricos en el dataset “Aseguradora.csv” los cuales son:

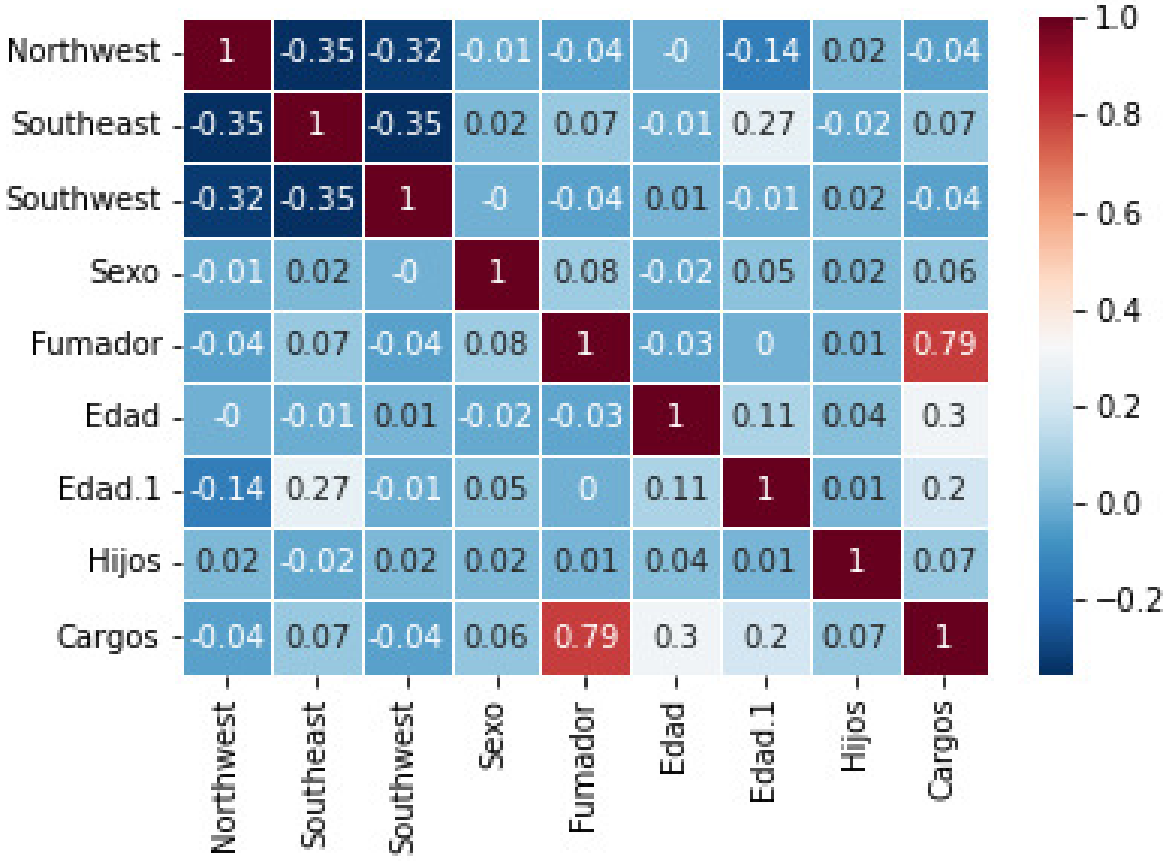
Campos	**Lista de categorías**
Sexo	[female, male]
Fumador	[Yes, No]
Región	[southwest, southeast, northeast, northwest]

Por lo tanto, los campos mencionados serán reemplazados por un total de 5 columnas:

Campo	**Tipo de valor**
Edad	Numérico
Female	Numérico
edad	Numérico
Hijos	Numérico
Fumador	Numérico
Southwest	Numérico
Southeast	Numérico
Northeast	Numérico
Cargos	**Numérico, valor a predecir**

Correlación de las variables

La relación entre cada campo y el valor a predecir, siguiendo el Coeficiente de correlación de Pearson es:



Clasificación

Se tiene un conjunto de datos, también llamado dataset, con 2000 registros y 21 campos.

Campo	Tipo de valor
Battery_power	Numérico
Bluetooth	Booleano
Clock_speed	Numérico
Dual_sim	Booleano
Fc_megapixel	Numérico

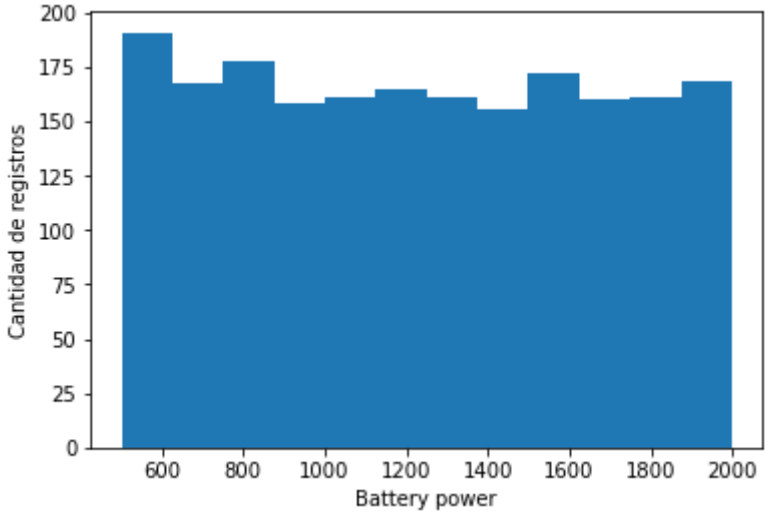
Four_g	Booleano
Int_memory	Numérico
M_depth	Numérico
Mobile_wt	Numérico
N_cores	Numérico
Pc_megapixel	Numérico
Px_height	Numérico
Px_width	Numérico
Ram	Numérico
Screen_height	Numérico
Screen_width	Numérico
Talk_time	Numérico
Three_g	Booleano
Touch_screen	Booleano
Wifi	Booleano
Price_range	**Categórico [0,1,2,3]**

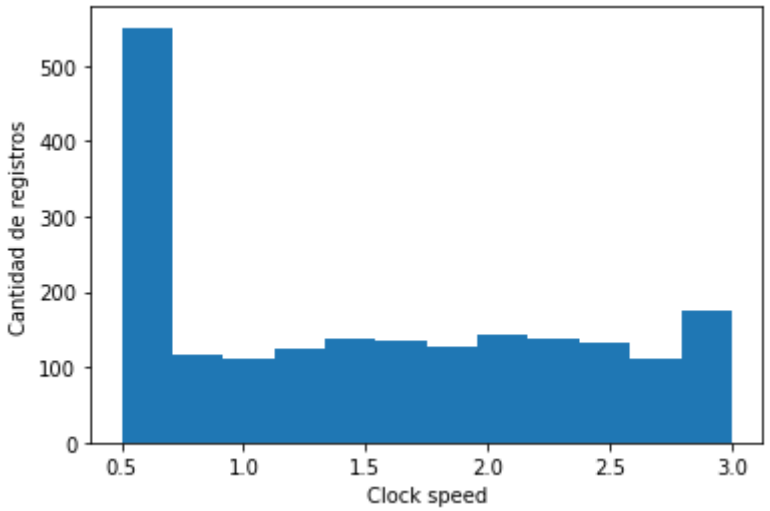
Campo	**Total**	**Media**	**Desviación standard**	**Mínimo**	**25%**	**50%**	**75%**	**Máximo**

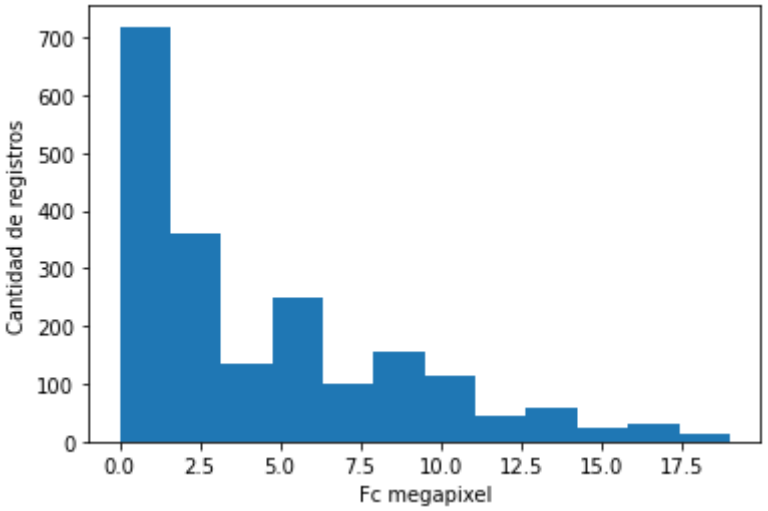
Battery_power	2000	1238.5	439.418	501	851.75	1226	1615.25	1998
Clock_speed	2000	1.52225	0.816004	0.5	0.7	1.5	2.2	3
Fc_megapixel	2000	4.3095	4.34144	0	1	3	7	19
Int_memory	2000	32.0465	18.1457	2	16	32	48	64
M_depth	2000	0.50175	0.288416	0.1	0.2	0.5	0.8	1
Mobile_wt	2000	140.249	35.3997	80	109	141	170	200
N_cores	2000	4.5205	2.28784	1	3	4	7	8
Pc_megapixel	2000	9.9165	6.06431	0	5	10	15	20
Px_height	2000	645.108	443.781	0	282.75	564	947.25	1960
Px_width	2000	1251.52	432.199	500	874.75	1247	1633	1998
Ram	2000	2124.21	1084.73	256	1207.5	2146.5	3064.5	3998
Screen_height	2000	12.3062	4.21325	5	9	12	16	19

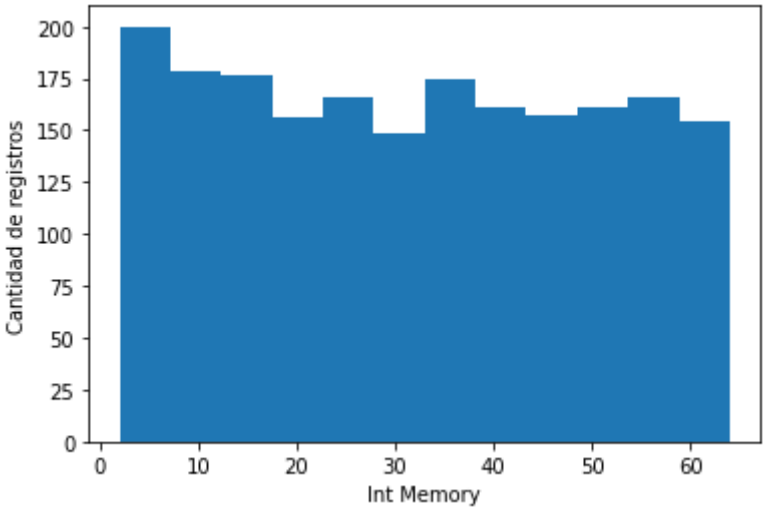
Screen_width	2000	5.767	4.3564	0	2	5	9	18
Talk_time	2000	11.011	5.46396	2	6	11	16	20

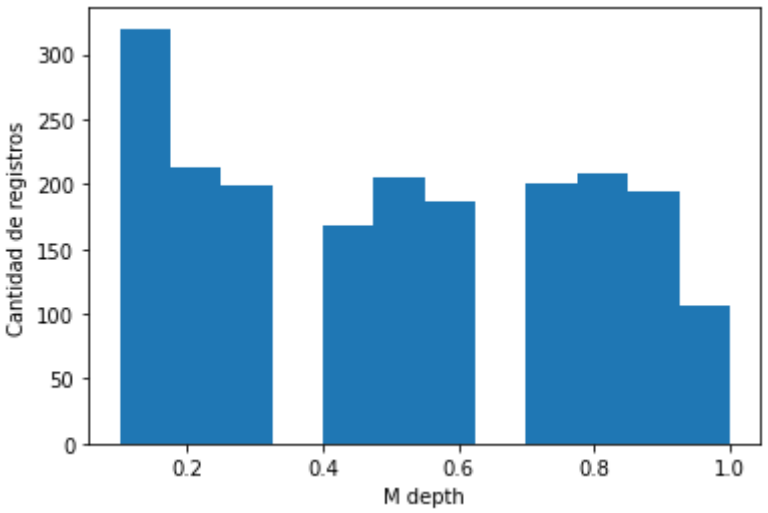
Distribución gráfica

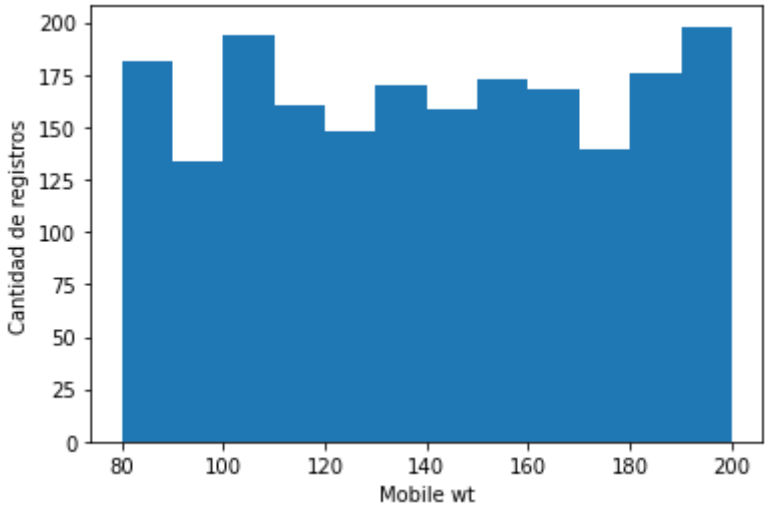


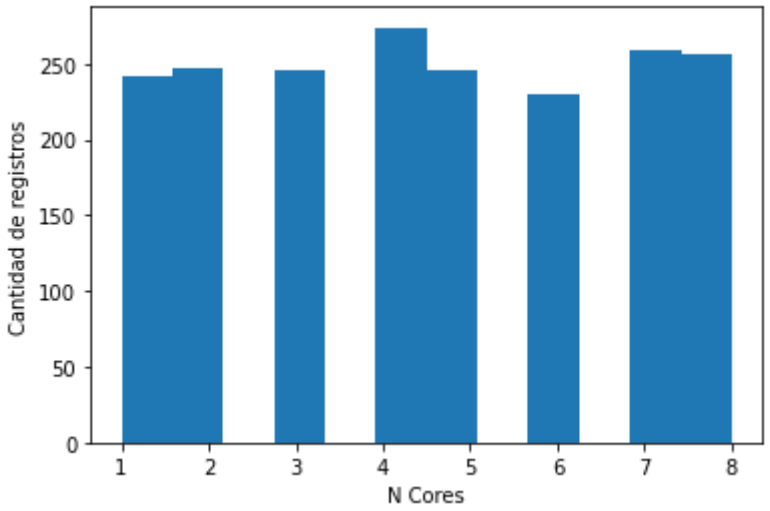


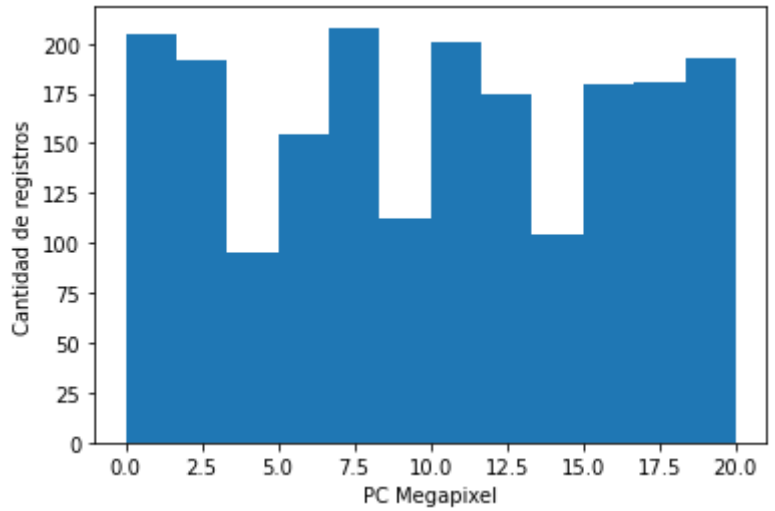

Cantidad de registros
0
100
200
300
400
500
600
700
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Fc megapixel


Cantidad de registros
0
25
50
75
100
125
150
175
200
0
10
20
30
40
50
60
Int Memory

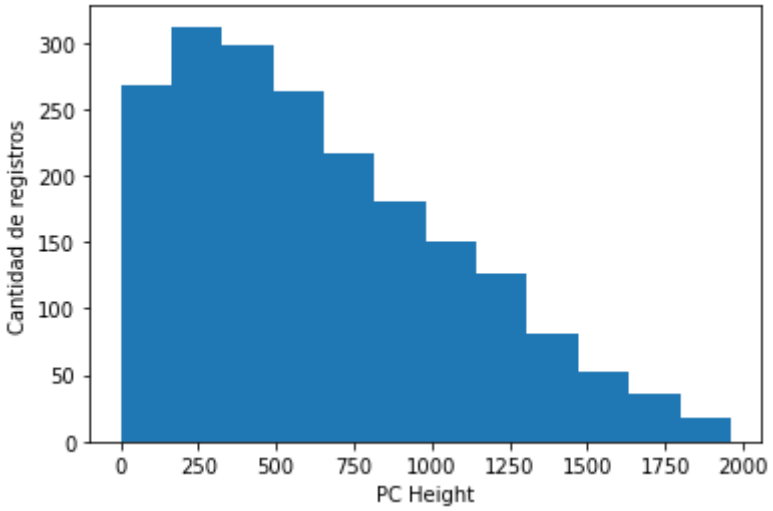

Cantidad de registros
0
50
100
150
200
250
300
0.2
0.4
0.6
0.8
1.0
M depth

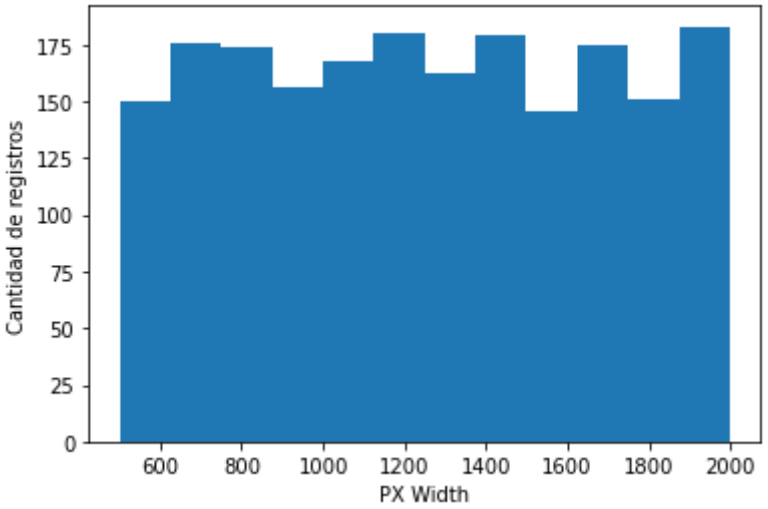

Cantidad de registros
0
25
50
75
100
125
150
175
200
80
100
120
140
160
180
200
Mobile wt

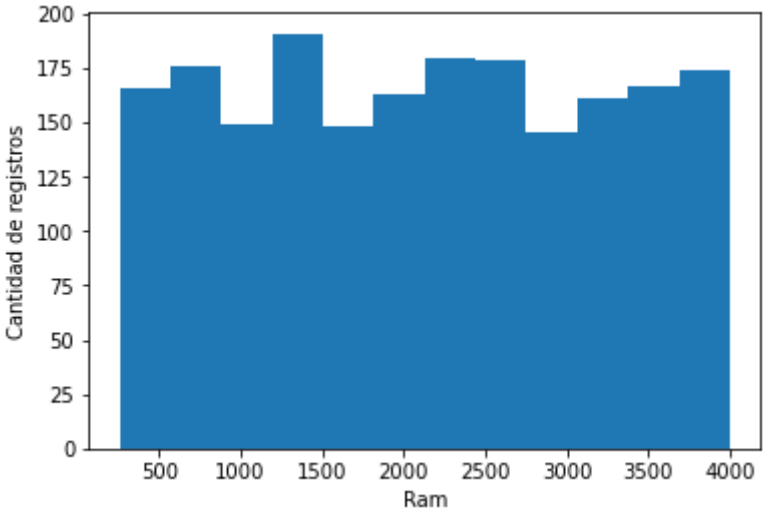

Cantidad de registros
0
50
100
150
200
250
1
2
3
4
5
6
7
8
N Cores

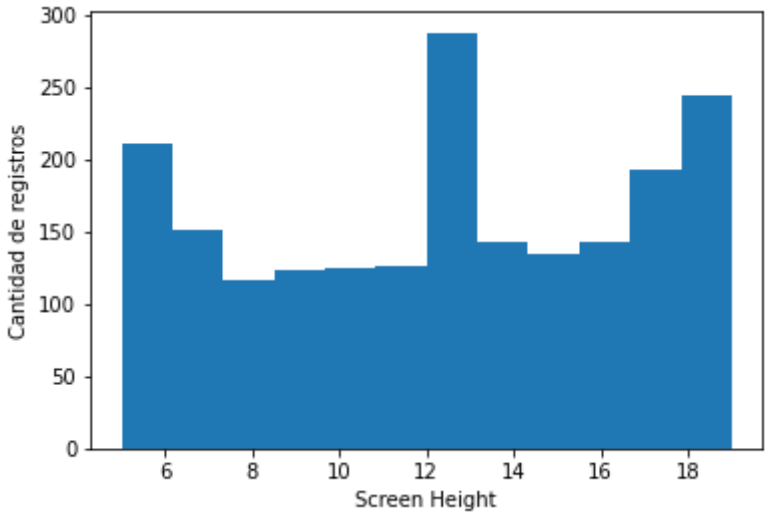

Cantidad de registros
0
25
50
75
100
125
150
175
200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
PC Megapixel

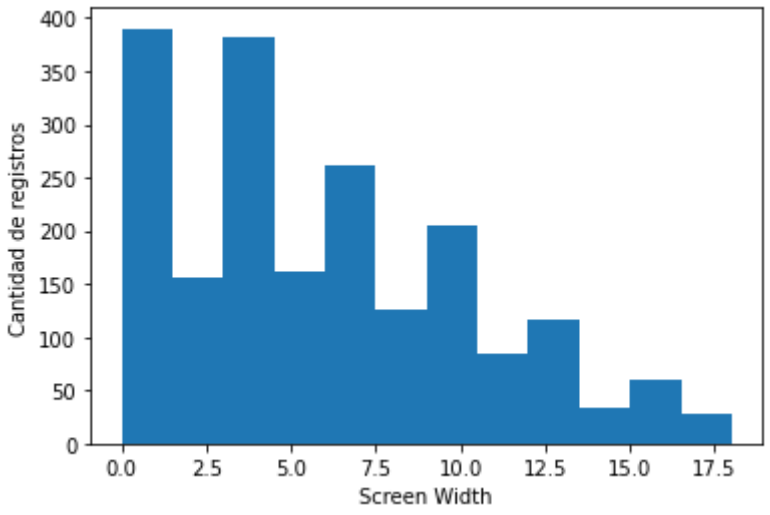
.

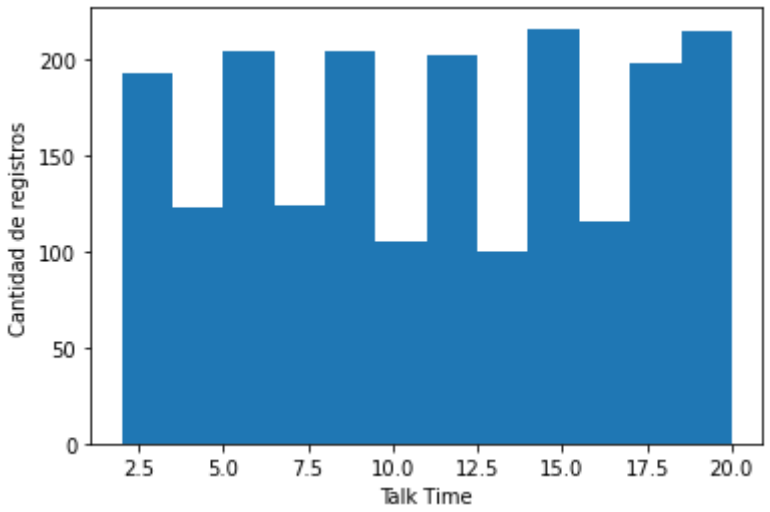

Cantidad de registros
300
250
200
150
100
50
0
0
250
500
750
1000
1250
1500
1750
2000
PC Height


Cantidad de registros
175
150
125
100
75
50
25
0
600
800
1000
1200
1400
1600
1800
2000
PX Width


Cantidad de registros
200
175
150
125
100
75
50
25
0
500
1000
1500
2000
2500
3000
3500
4000
Ram


Cantidad de registros
Screen Height


Cantidad de registros
Screen Width


Cantidad de registros
Talk Time

Aplicación de las variables dummy

Existen cuatro campos categóricos en el dataset “Celulares.csv” los cuales son: Bluetooth, Four_g, Three_g, touch_screen, wifi. Sin embargo, al ser todos con categorías binarias variando entre las categorías 0 y 1, no es necesario aplicar el proceso de las variables dummy.

Campos	Lista de categorías
Bluetooth	[female, male]
Four_g	[0,1]
Three_g	[0,1]
touch_screen	[0,1]
Wifi	[0,1]

Correlación de las variables

Al tener demasiados campos, y para facilitar la legibilidad de la información, en este ejercicio se optó por una tabla de dos campos en vez de un mapa de color de profundidad como en el ejercicio de regresión.

La relación entre cada campo y el valor a predecir, siguiendo el Coeficiente de correlación de Pearson es:

Campo	Coeficiente de correlación de Pearson en relación al campo a predecir
Ram	0.917046
Battery_power	0.200723
Px_width	0.165818
Px_height	0.148858

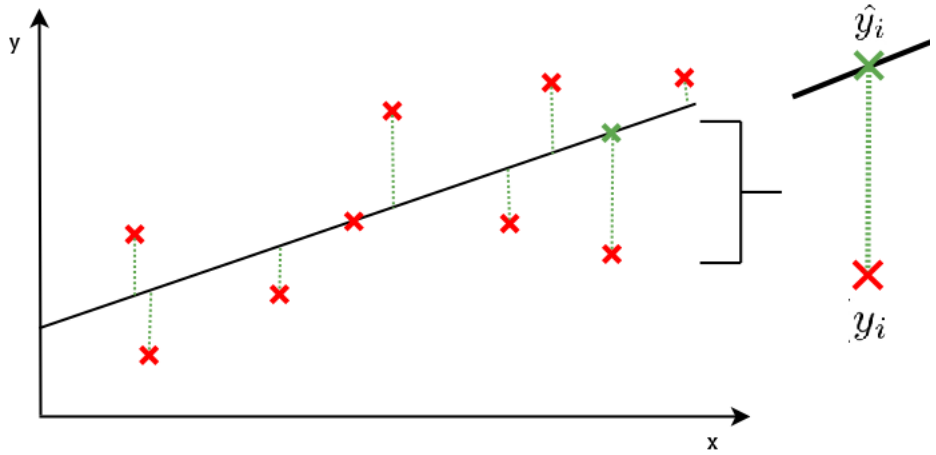
Int_memory	0.044435
Screen_width	0.0387113
Pc_megapixel	0.0335993
Three_g	0.0236112
Screen_height	0.0229861
Fc_megapixel	0.0219982
Talk_time	0.0218589
Bluetooh	0.0205729
Wifi	0.0187848
Dual_sim	0.0174445
Four_g	0.0147717
N_cores	0.00439927
M_depth	0.000853037
Clock_speed	-0.00660569
Mobile_wt	-0.0303022
Touch_screen	-0.0304111

5. Aplicación

Método de evaluación

El método utilizado para evaluar la eficiencia de los modelos es el factor R cuadrado (R-squared en inglés).

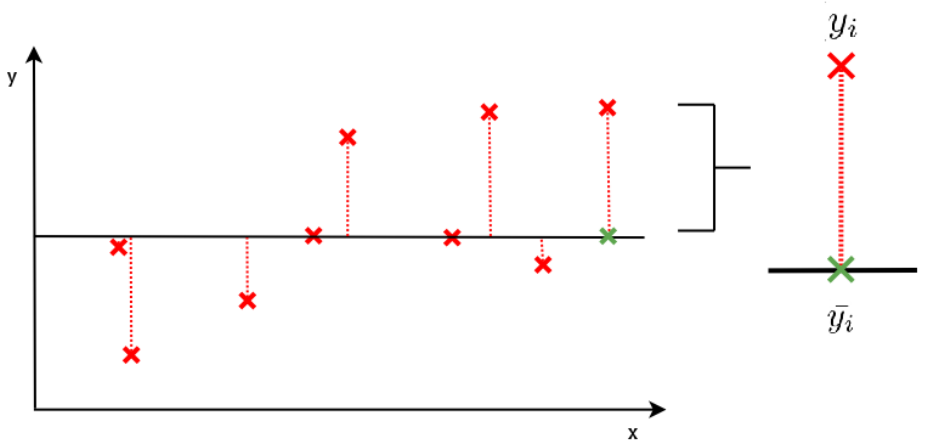
Para entender el concepto, se parte de un ejemplo de dos dimensiones en el que se tiene un modelo de regresión lineal.



Se le denomina residuo a la diferencia entre el valor real y el predicho. Para calcular R cuadrado, es necesario obtener la Suma de los Cuadrados Residuales.

$$SS_{res} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Para el mismo ejemplo, se puede trazar un promedio entre el mínimo valor y máximo y obtener la Suma de los Cuadrados Totales, el cual es invariable.



$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y}_i)^2$$

Para obtener R cuadrado es necesario tener ambas sumatorias, las cuales son utilizadas en la siguiente ecuación:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Cuanto más cerca sea el valor final a 1, más eficiente es el modelo.

Optimización de los modelos

Para encontrar la mejor configuración de los hiperparámetros en cada modelo, independientemente de si se tratan de clasificación o regresión, es con GridSearchCV, que es un método de la librería sklearn que realiza una búsqueda exhaustiva sobre los mejores parámetros a la hora de configurar cada modelo.

Los dos parámetros principales que recibe son: *estimator*, que es el modelo que se desea optimizar, y *param_grid* que recibe un objeto con una serie de listas de los parámetros que explorará para encontrar el mejor resultado (por ejemplo, para la creación de un bosque aleatorio, se puede marcar que pruebe buscando la combinación entre 1 y 100 árboles).

Ejercicio de regresión

Consiste en predecir la cantidad que pagará un individuo a la aseguradora en base a sus datos personales. Para esto, se cuenta con un documento llamado “Aseguradora.csv” que contiene un total de 1338 registros con un total de 8 campos cada uno, los cuales son:

Campo	**Tipo de valor**
Edad	Numérico
Sexo	String
Edad	Numérico

Hijos	Numérico
Fumador	String
Región	String
Cargos	**Numérico, valor a predecir**

Regresión Lineal Múltiple

1. Se carga el dataset a través del archivo "Aseguradora.csv".

```
# Cómo importar las librerías
import numpy as np
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Aseguradora.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 6].values
```

3. Se realiza el preprocesado de los datos añadiendo las variables dummy.

```
# Variables Dummy
labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder = make_column_transformer((OneHotEncoder(),
[4]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
```

```
onehotencoder = make_column_transformer((OneHotEncoder(),
[2]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 5] = labelencoder_X.fit_transform(X[:,5 ])
onehotencoder = make_column_transformer((OneHotEncoder(),
[5]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]
```

4. Se realiza la eliminación hacia atrás:
    a. Paso 1: se elige un nivel de significación (SL en inglés) para permanecer en el modelo. Con esto se mide la relevancia de la variable en el algoritmo.
    b. Paso 2: se calcula el modelo con todas las posibles variables predictoras (All-in).
    c. Paso 3: se considera la variable predictora con el p-valor más grande. Si ésta supera el nivel de significación, se avanza al siguiente paso. Caso contrario, se finaliza el proceso.
    d. Paso 4: se elimina la variable predictora.
    e. Paso 5: se vuelve a ajustar el modelo desde el paso 3 hasta que todas las variables tengan un nivel de significación menor al asignado en el paso 1.

```
# Construir el modelo óptimo de RLM utilizando la Eliminación
hacia atrás
import statsmodels.api as sm

def backwardElimination(x, sl):
    numVars = len(x[0])
```

```
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x.tolist()).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) ==
maxVar):
                    x = np.delete(x, j, 1)
    return x

SL = 0.05
X_opt = X[:, [ 0,1, 2, 3, 4, 5, 6]]
X_Modeled = backwardElimination(X_opt, SL)
```

5. Se dividen los datos de entrenamiento y de test.

```
# Dividir el data set en conjunto de entrenamiento y conjunto
de testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_Modeled,
y, test_size = 0.25, random_state = 0)
```

6. Se entrena el modelo con el dataset de entrenamiento.

```
# Ajustar el modelo de Regresión lineal múltiple con el
conjunto de entrenamiento
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)
```

7. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
y_pred = regression.predict(X_test)
```

8. Se calcula el valor $\mathbf{R^2}$ para cuantificar el nivel de predicción.

```
#Obtenemos R cuadrado
from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Resultado final R^2:0.799

Árbol de decisión

1. Se carga el dataset a través del archivo "Aseguradora.csv".

```
# Cómo importar las librerías
import numpy as np
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Aseguradora.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 6].values
```

3. Se realiza el preprocesado de los datos añadiendo las variables dummy.

```
# Variables Dummy
labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder    =    make_column_transformer((OneHotEncoder(),
[4]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
onehotencoder    =    make_column_transformer((OneHotEncoder(),
[2]), remainder = "passthrough")
```

```
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 5] = labelencoder_X.fit_transform(X[:,5 ])
onehotencoder    =    make_column_transformer((OneHotEncoder(),
[5]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]
```

4. Se divide el dataset en un conjunto de entrenamiento y uno de test.

```
# Dividir el dataset en conjunto de entrenamiento y conjunto
de testing
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,
test_size = .25, random_state = 0)
```

5. Se utiliza GridSearchCV para buscar la mejor configuración para el árbol en base a la lista de parámetros dentro de param_grid.

```
#Validación
from sklearn.model_selection import GridSearchCV
#Validación
param_grid = {"criterion": ["mse", "mae"],
              "min_samples_split": [10, 20, 40],
              "max_depth": [2, 6, 8],
              "min_samples_leaf": [20, 40, 100],
              "max_leaf_nodes": [5, 20, 100],
              }
regression = DecisionTreeRegressor( random_state = 0)
```

```
regression.fit(X_train,y_train)
grid_cv_dtm = GridSearchCV(regression, param_grid, cv=5)
grid_cv_dtm.fit(X,y)
best_params = grid_cv_dtm.best_params_
```

6. Se vuelve a probar el modelo con la configuración retornada en la variable best_params.

```
regression = DecisionTreeRegressor( criterion= 'mse',
max_depth=6, max_leaf_nodes=20, min_samples_leaf=20,
min_samples_split=10, random_state = 0)
regression.fit(X_train,y_train)
```

7. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
y_pred = regression.predict(X_test)
```

8. Se calcula el valor R2 para cuantificar el nivel de predicción.

```
#Obtenemos R cuadrado
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
```

Resultado final R^2:0.893

Bosques aleatorios

1. Se carga el dataset a través del archivo “Aseguradora.csv”.

```
# Cómo importar las librerías
import numpy as np
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Aseguradora.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 6].values
```

3. Se realiza el preprocesado de los datos añadiendo las variables dummy.

```
# Codificar datos categóricos
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import make_column_transformer

labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder = make_column_transformer((OneHotEncoder(),
[4]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
onehotencoder = make_column_transformer((OneHotEncoder(),
[2]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 5] = labelencoder_X.fit_transform(X[:,5 ])
onehotencoder = make_column_transformer((OneHotEncoder(),
[5]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]
```

4. Se dividen los datos para el conjunto de entrenamiento y de test.

```
# Dividir el data set en conjunto de entrenamiento y conjunto de testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

5. Se utiliza GridSearchCV para buscar la mejor configuración para el bosque aleatorio en base a la lista de parámetros dentro de param_grid.

```
##Validación
from sklearn.model_selection import GridSearchCV
param_grid = {'n_estimators': range(1, 100),'max_features': ['auto', 'sqrt', 'log2'],'max_depth' : [4,5,6,7,8]}

rfr=RandomForestRegressor(random_state=0)
CV_rfr = GridSearchCV(estimator=rfr, param_grid=param_grid, cv= 5)
CV_rfr.fit(X_train, y_train)
best_params = CV_rfr.best_params_
```

6. Se vuelve a probar el modelo con la configuración retornada en la variable best_params.

```
rfc1=RandomForestRegressor(random_state=0,
max_features='auto', n_estimators= 67, max_depth=4)
rfc1.fit(X_train, y_train)
```

7. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
#Predicción
y_pred=rfc1.predict(X_test)
```

8. Se calcula el valor R2 para cuantificar el nivel de predicción.

```
#Porcentaje de error
```

```
from sklearn.metrics import r2_score
r2=r2_score(y_test, y_pred)
```

Resultado final R^2:0.900

Red Neuronal Artificial

1. Se carga el dataset a través del archivo "Aseguradora.csv".

```
# Cómo importar las librerías
import numpy as np
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Aseguradora.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 6].values
```

3. Se realiza el preprocesado de los datos añadiendo las variables dummy.

```
# Codificar datos categóricos
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import make_column_transformer

labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder = make_column_transformer((OneHotEncoder(),
[4]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]
```

```
X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
onehotencoder = make_column_transformer((OneHotEncoder(),
[2]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 5] = labelencoder_X.fit_transform(X[:,5 ])
onehotencoder = make_column_transformer((OneHotEncoder(),
[5]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]
```

4. Se dividen los datos para el conjunto de entrenamiento y de test.

```
# Dividir el data set en conjunto de entrenamiento y conjunto
de testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.25, random_state = 0)
```

5. Se define el modelo de la red neuronal.

```
#Modelo
def larger_model():
    # create model
    model = Sequential()
                        model.add(Dense(20,      input_dim=8,
kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
```

```
    return model
```

6. Se utiliza GridSearchCV para buscar la mejor configuración para la red neuronal en base a la lista de parámetros dentro de param_grid.

```
#Validación
from sklearn.model_selection import GridSearchCV
model = KerasRegressor(build_fn=larger_model)

param_grid = { 'batch_size':[5,10,20],
'epochs':[100,1000,2000]}
grid = GridSearchCV(estimator=model, param_grid=param_grid,
n_jobs=1)
grid.fit(X_train, y_train)
best_params = grid.best_params_
```

7. Se prueba el modelo con la configuración retornada en la variable best_params.

```
estimator = KerasRegressor(build_fn=larger_model, epochs=2000,
batch_size=5, verbose=0)
estimator.fit(X_train, y_train)
```

8. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
#Predicción
y_pred=rfc1.predict(X_test)
```

9. Se calcula el valor R2 para cuantificar el nivel de predicción.

```
#Porcentaje de error
from sklearn.metrics import r2_score
r2=r2_score(y_test, y_pred)
```

Resultado final R^2:0.796

Comparativa

Resultados (primeros 15 registros):

Valor Real	Regresión Lineal Múltiple	Árbol de decisión	Bosques aleatorios	Red Neuronal Artificial
9724.53	11169.9	11766.5	12547	10643.6
8547.69	9486.71	10853.7	10778.3	9797.64
45702	38181.1	44047.5	45050.7	36416.3
12950.1	16266.3	13950.9	14365.2	14605.7
9644.25	6914.65	10853.7	11410.8	7962.72
4500.34	3963.48	6903.45	6477.96	5510.3
2198.19	1579.4	3351.65	3071.35	3242.1
11436.7	14385.3	10853.7	11948.7	13173.2
7537.16	9012.58	6903.45	7186.05	8704.99
5425.02	7508.46	5166.16	6405.47	7115.43
6753.04	4491.77	6903.45	6845.81	6332.36
10493.9	10279.6	10853.7	12048.5	10195.9
7337.75	8801.3	10853.7	8877.3	8750.98
4185.1	3798.02	6903.45	6223.78	5185.12
18310.7	27926.2	18769.7	20000	27875.9

Modelo	R^2
Bosques aleatorios	0.900
Árbol de decisión	0.893
Regresión lineal múltiple	0.799
Red Neuronal Artificial	0.796

Como se ve en la comparativa de resultados, el modelo de Bosques aleatorios es el que ofreció los mejores resultados con la configuración presentada. Es importante tener en cuenta que si se optimizan aún los modelos para encontrar la mejor configuración posible, puede que la tabla de posición se vea modificada por lo que el presente trabajo no indica que un modelo sea superior a otro per se. Cabe recordar que R^2 indica la precisión del modelo y que cuanto más cercano a 1 sea el valor, mayor exactitud es la del modelo en base a los datos de prueba.

Ejercicio de clasificación

Consiste en clasificar el precio de los celulares en 4 categorías, de 0 a 3, que representan de forma ascendente, el rango del precio final. Para esto, se cuenta con un documento "Celulares.csv" que contiene 2000 registros de especificaciones técnicas más la categoría a la que pertenecen cada celular.

Cada registro cuenta con los siguientes 21 campos:

Campo	Tipo de valor
Battery_power	Numérico
Bluetooth	Booleano
Clock_speed	Numérico

Dual_sim	Booleano
Fc_megapixel	Numérico
Four_g	Booleano
Int_memory	Numérico
M_depth	Numérico
Mobile_wt	Numérico
N_cores	Numérico
Pc_megapixel	Numérico
Px_height	Numérico
Px_width	Numérico
Ram	Numérico
Screen_height	Numérico
Screen_width	Numérico
Talk_time	Numérico
Three_g	Booleano
Touch_screen	Booleano
Wifi	Booleano
Price_range	**Categórico [0,1,2,3]**

K-NN

1. Se carga el dataset a través del archivo “Celulares.csv”.

```
# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Celulares.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, 20].values
```

3. Se dividen los datos de entrenamiento y de test.

```
# Dividir el data set en conjunto de entrenamiento y conjunto de testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .25, random_state = 0)
```

4. Se utiliza GridSearchCV para buscar la mejor configuración para el algoritmo en base a la lista de parámetros dentro de param_grid.

```
#Se busca el mejor modelo
knn = KNeighborsClassifier()
from sklearn.model_selection import GridSearchCV
k_range = list(range(1, 101))
param_grid = dict(n_neighbors=k_range)

grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy', verbose=1)
grid.fit(x_train, y_train)
best_params = grid.best_params_
```

5. Se vuelve a probar el modelo con la configuración retornada en la variable best_params.

```
# Ajustar el clasificador en el Conjunto de Entrenamiento
from sklearn.neighbors import KNeighborsClassifier
```

```
classifier = KNeighborsClassifier(n_neighbors=12, metric = "minkowski",
p = 2)
classifier.fit(X_train, y_train)
```

6. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
# Predicción de los resultados con el Conjunto de Testing
y_pred  = classifier.predict(X_test)
```

7. Se calcula el valor R2 para cuantificar el nivel de predicción.

```
#R2
from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Resultado final R^2:0.961

Árboles de decisión

1. Se carga el dataset a través del archivo “Celulares.csv”.

```
# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Celulares.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, 20].values
```

3. Se dividen los datos de entrenamiento y test.

```
# Dividir el data set en conjunto de entrenamiento y conjunto de
testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.25, random_state = 0)
```

4. Se utiliza GridSearchCV para buscar la mejor configuración para el árbol en base a la lista de parámetros dentro de param_grid.

```
#Se busca el mejor modelo
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
from sklearn.model_selection import GridSearchCV
param_grid ={'criterion':['gini','entropy'],'max_depth':[range(1,101)]}

grid = GridSearchCV(classifier, param_grid, cv=10)
grid.fit(x_train, y_train)
best_params = grid.best_params_
```

5. Se vuelve a probar el modelo con la configuración retornada en la variable best_params.

```
# Ajustar el clasificador de Árbol de Decisión en el Conjunto de Entrenamiento
classifier = DecisionTreeClassifier(criterion= "entropy", max_depth=67, random_state = 0)
classifier.fit(X_train, y_train)
```

6. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
# Predicción de los resultados con el Conjunto de Testing
y_pred  = classifier.predict(X_test)
```

7. Se calcula el valor R2 para cuantificar el nivel de predicción.

```
from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Resultado final R^2:0.892

Bosques aleatorios

1. Se carga el dataset a través del archivo “Celulares.csv”.

```
# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Importar el data set
dataset = pd.read_csv('Celulares.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, 20].values
```

3. Se dividen los datos de entrenamiento y test.

```
# Dividir el data set en conjunto de entrenamiento y conjunto de testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

4. Se utiliza GridSearchCV para buscar la mejor configuración para el bosque en base a la lista de parámetros dentro de param_grid.

```
#Se busca el mejor modelo
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
from sklearn.model_selection import GridSearchCV
k_range = list(range(1, 101))
param_grid = dict(n_estimators=k_range, criterion=['entropy','gini'])

grid = GridSearchCV(classifier, param_grid, cv=10, )
grid.fit(x_train, y_train)
best_params = grid.best_params_
```

5. Se vuelve a probar el modelo con la configuración retornada en la variable best_params.

```
# Ajustar el clasificador Random Forest en el Conjunto de Entrenamiento
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 67, criterion = "entropy", random_state = 0)
classifier.fit(X_train, y_train)
```

6. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
# Predicción de los resultados con el Conjunto de Testing
y_pred  = classifier.predict(X_test)
```

7. Se calcula el valor R2 para cuantificar el nivel de predicción.

```
from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Resultado final R^2:0.906

Redes Neuronales Artificiales

1. Se carga el dataset a través del archivo "Celulares.csv".

```
# Librerías
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

# Cargar dataset

dataset = pandas.read_csv('Celulares.csv')
```

2. Se divide el dataset entre los valores predictores y el valor a predecir.

```
X = dataset.iloc[:,:-1].values
Y = dataset.iloc[:, 20].values
```

3. Se crea el modelo de la red.

```
# MCreación del modelo
def baseline_model():
    model = Sequential()
    model.add(Dense(40, input_dim=20, activation='relu'))
    model.add(Dense(4, activation='softmax'))
       model.compile(loss='categorical_crossentropy',  optimizer='adam',
metrics=['accuracy'])
```

```
    return model
```

4. Se dividen los datos de entrenamiento y de test.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.2, random_state = 0);
```

5. Se utiliza GridSearchCV para buscar la mejor configuración para el árbol en base a la lista de parámetros dentro de param_grid

```
#Se busca el mejor modelo
from sklearn.model_selection import GridSearchCV
param_grid ={'batch_size':[5,10,20],'epochs':[range(10,100,500)]}

grid= GridSearchCV(baseline_model, param_grid, cv=10)
grid.fit(X_train,y_train)
best_params = grid.best_params_
```

6. Se vuelve a probar el modelo con la configuración retornada en la variable best_params.

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
batch_size=10, verbose=0)
estimator.fit(X_train, y_train)
```

7. Se valida el rendimiento prediciendo los datos de test con el modelo.

```
# Predicción de nuestros modelos
y_pred  = estimator.predict(X_test)
```

8. Se calcula el valor R2 para cuantificar el nivel de predicción.

```
from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Resultado final R^2:0.961

Comparativa

Resultados (primeros 15 registros):

Valor Real	K-NN	Árbol de decisión	Bosques aleatorios	Red Neuronal Artificial
3	3	3	3	3
0	0	0	0	0
2	2	2	2	2
2	2	2	2	2
2	2	2	2	2
0	0	0	0	0
0	0	0	0	0
3	3	3	3	3
3	3	3	3	3
1	1	1	1	1
1	0	0	1	1
3	3	3	3	3
0	0	0	0	0
2	2	2	2	2
3	3	3	3	3

Modelo	R^2
Red Neuronal Artificial	0.961
K-NN	0.941
Bosques aleatorios	0.906
Árbol de decisión	0.892

Como se ve en la comparativa de resultados, el modelo de Redes Neuronales Artificiales es el que ofreció los mejores resultados con la configuración presentada con un valor R^2 de 0.961.

Conclusión

Según lo indicado al inicio, los objetivos planteados por el presente trabajo fueron: investigar las bases de Machine Learning y aplicar su conocimiento para resolver dos ejercicios de predicción supervisada. Finalizado el trabajo, se llegó a las siguientes conclusiones:

- El trabajo cumplió su objetivo de ofrecer una introducción al área de la Inteligencia Artificial.
- Se logró transmitir las bases de Machine Learning. Se la definió como un campo de investigación para lograr que las máquinas sean capaces de aprender mediante distintos tipos de aprendizaje (supervisado, no supervisado y por refuerzo).
- Se investigó y aplicó parte de los algoritmos existentes para la resolución de los ejercicios planteados durante el trabajo, tanto modelos de regresión como clasificación. En ambos ejercicios, se llegó en todos los modelos una predicción mayor al 80%.
- Quedó demostrada la importancia del análisis de datos y su limpieza previo a la ejecución del modelo para conseguir un mejor resultado.
- Por último, el trabajo cumple su función de ser una introducción y guía para la comprensión del campo de Machine Learning y su aplicación en un ámbito real.

El conocimiento demostrado durante el trabajo puede ser aplicado en diversos campos, sin importar si están relacionados al ámbito de la informática. Como se demostró en los ejercicios, los modelos pueden ser de ayuda para empresas de seguros así como una tienda de celulares. Yendo más a fondo y dependiendo de los datos con los que uno puede contar, su ejecución puede realizarse para predecir la posibilidad de éxito de un grupo de estudiantes para un examen; predecir la probabilidad de que un paciente presente una enfermedad en base a sus registros médicos; o predecir qué representante elegirá una región durante una votación presidencial; entre muchos posibles casos.

También cabe aclarar la importancia de un equipo tecnológico capaz de procesar estos modelos y cómo a mayor capacidad, se puede aspirar a una mayor precisión en la ejecución del proceso. Por ejemplo, durante la aplicación de las redes neuronales, la cantidad de epochs puede ascender hasta los 10000-100000 durante la fase de entrenamiento si el equipo es capaz de ejecutar dicho entrenamiento. Por lo que aspirar a una mejora a nivel tecnológico puede aumentar las probabilidades de éxito de los modelos existentes e incluso, podría ser capaz de llevar a la creación de nuevos modelos.

Bibliografía

- AA VV, (Consulta: 20/04/2022), “¿Qué es el aprendizaje supervisado?”, <https://www.tibco.com/reference-center/what-is-supervised-learning>
- AA VV, (2019), “Redes neuronales”, <https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb>
- Amat Rodrigo, Joaquin, (2020), “Regularización”, <https://www.cienciadedatos.net/documentos/31_seleccion_de_predictores_subset_selection_ridge_lasso_dimension_reduction>
- Andrews, Heater, (2019), “Aprendizaje por refuerzo”, <https://www.kudaw.com/blog/c%C3%B3mo-las-m%C3%A1quinas-aprenden-desde-cero>
- Lopez Briega, Raul, (2017), "Introducción a la Inteligencia Artificial", <https://relopezbriega.github.io/blog/2017/06/05/introduccion-a-la-inteligencia-artificial/>

- Martinez, Jesús, (2019), "¿Qué es overfitting?", <https://datasmarts.net/es/que-es-overfitting-y-como-lo-detectamos/>
- Martinez, Jesús, (2019), "¿Qué es underfitting?" ,<https://datasmarts.net/es/que-es-underfitting-y-como-lo-detectamos/>
- Martinez Heras, Jose, (2020), "Error cuadrático medio", <https://www.iartificial.net/error-cuadratico-medio-para-regresion/>
- Ng, Ritchie, (2017), "Funcionalidad de GridSearch", <https://www.ritchieng.com/machine-learning-efficiently-search-tuning-param/>
- Nielsen, Michael, (2019), "Propagación hacia atrás", <http://neuralnetworksanddeeplearning.com/chap2.html>

Anexo

Código fuente de ejercicio de regresión

Regresión Lineal Múltiple

```
# Regresión Lineal Múltiple

# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Aseguradora.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 6].values

# Codificar datos categóricos
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import make_column_transformer

labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder = make_column_transformer((OneHotEncoder(), [4]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
onehotencoder = make_column_transformer((OneHotEncoder(), [2]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 5] = labelencoder_X.fit_transform(X[:,5 ])
onehotencoder = make_column_transformer((OneHotEncoder(), [5]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

# Dividir el dataset en conjunto de entrenamiento y conjunto de
testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2, random_state = 0)
```

```
# Ajustar el modelo de Regresión lineal múltiple con el conjunto de
entrenamiento
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)

# Construir el modelo óptimo de RLM utilizando la Eliminación hacia
atrás
import statsmodels.api as sm
X = np.append(arr = np.ones((1338,1)).astype(int), values = X, axis
= 1)
SL = 0.05

import statsmodels.api as sm
def backwardElimination(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x.tolist()).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) ==
maxVar):
                    x = np.delete(x, j, 1)
    return x

SL = 0.05
X_opt = X[:, [0, 1, 2, 3, 4, 5, 6]]
X_Modeled = backwardElimination(X_opt, SL)

#Obtenemos R cuadrado
from sklearn.metrics import r2_score
y_pred = regression.predict(X_test)
r2= r2_score(y_test, y_pred)
```

Árbol de decisión

```
# Regresión con Árboles de Decisión

# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Aseguradora.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 6].values

# Codificar datos categóricos
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import make_column_transformer

labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder = make_column_transformer((OneHotEncoder(), [4]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)
# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
onehotencoder = make_column_transformer((OneHotEncoder(), [2]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)
# Evitar la trampa de las variables ficticias
X = X[:, 1:]
```

```
X[:, 5] = labelencoder_X.fit_transform(X[:,5 ])
onehotencoder = make_column_transformer((OneHotEncoder(), [5]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

# Dividir el data set en conjunto de entrenamiento y conjunto de
testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= .25, random_state = 0)

# Ajustar la regresión con el dataset
from sklearn.tree import DecisionTreeRegressor
regression = DecisionTreeRegressor( random_state = 0)
regression.fit(X_train,y_train)

y_pred = regression.predict(X_test)

import statsmodels.api as sm
regressor_OLS_test = sm.OLS(y_pred, X_test.tolist()).fit()
print(regressor_OLS_test.summary())
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
```

Bosques aleatorios

```
# Regresión Bosques Aleatorios

# Cómo importar las librerías
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Aseguradora.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 6].values

# Codificar datos categóricos
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import make_column_transformer

labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder = make_column_transformer((OneHotEncoder(), [4]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
onehotencoder = make_column_transformer((OneHotEncoder(), [2]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 5] = labelencoder_X.fit_transform(X[:,5 ])
onehotencoder = make_column_transformer((OneHotEncoder(), [5]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)
```

```
# Evitar la trampa de las variables ficticias
X = X[:, 1:]

# Dividir el data set en conjunto de entrenamiento y conjunto de
testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.25, random_state = 0)

# Ajustar el Random Forest con el dataset
from sklearn.ensemble import RandomForestRegressor
regression = RandomForestRegressor(n_estimators = 1000, random_state
= 0)
regression.fit(X, y)

# Predicción de nuestros modelos con Random Forest
y_pred = regression.predict(X_test)

import statsmodels.api as sm
regressor_OLS_test = sm.OLS(y_pred, X_test.tolist()).fit()
print(regressor_OLS_test.summary())

# Predicción de nuestros modelos
y_pred = regression.predict(X_test)

from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Red Neuronal Artificial

```
# RNA- Regression

# Cómo importar las librerías
from pandas import read_csv
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Importar el data set
dataset = read_csv('Aseguradora.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 6].values

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import make_column_transformer

labelencoder_X = LabelEncoder()
X[:, 4] = labelencoder_X.fit_transform(X[:, 4])
onehotencoder = make_column_transformer((OneHotEncoder(), [4]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
onehotencoder = make_column_transformer((OneHotEncoder(), [2]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

X[:, 5] = labelencoder_X.fit_transform(X[:, 5])
```

```
onehotencoder = make_column_transformer((OneHotEncoder(), [5]),
remainder = "passthrough")
X = onehotencoder.fit_transform(X)

# Evitar la trampa de las variables ficticias
X = X[:, 1:]

#Modelo
def larger_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=8, kernel_initializer='normal',
activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

estimators = []
estimators.append(('standardize', StandardScaler()))

from sklearn.model_selection import train_test_split
import numpy as np

X = np.asarray(X).astype(np.float32)
estimator = KerasRegressor(build_fn=larger_model, epochs=10000,
batch_size=5, verbose=0)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
= 0.2, random_state = 0);
estimator.fit(X_train, y_train)

# Predicción de nuestros modelos
y_pred  = estimator.predict(X_test)
from sklearn.metrics import r2_score
```

```
r2 =r2_score(y_test, y_pred)
```

Código fuente de ejercicio de clasificación

K-NN

```
# K - Nearest Neighbors (K-NN)

# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Celulares.csv')
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, 20].values

# Codificar datos categóricos
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import make_column_transformer

# Dividir el data set en conjunto de entrenamiento y conjunto de
testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= .25, random_state = 0)

# Ajustar el clasificador en el Conjunto de Entrenamiento
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors =5, metric =
"minkowski", p = 2)
```

```
classifier.fit(X_train, y_train)

# Predicción de los resultados con el Conjunto de Testing
y_pred  = classifier.predict(X_test)

# Elaborar una matriz de confusión
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

#R2
from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Árboles de decisión

```
# Clasificación con árboles de Decisión

# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Celulares.csv')
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, 20].values

# Dividir el data set en conjunto de entrenamiento y conjunto de
testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.25, random_state = 0)
```

```
# Ajustar el clasificador de Árbol de Decisión en el Conjunto de
Entrenamiento
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = "entropy",
random_state = 0)
classifier.fit(X_train, y_train)

# Predicción de los resultados con el Conjunto de Testing
y_pred  = classifier.predict(X_test)

# Elaborar una matriz de confusión
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Bosques aleatorios

```
# Random Forest Classification

# Cómo importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el data set
dataset = pd.read_csv('Celulares.csv')

X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, 20].values
```

```
# Dividir el data set en conjunto de entrenamiento y conjunto de testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Ajustar el clasificador  Random Forest en el Conjunto de Entrenamiento
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10000, criterion = "entropy", random_state = 0)
classifier.fit(X_train, y_train)

# Predicción de los resultados con el Conjunto de Testing
y_pred  = classifier.predict(X_test)

# Elaborar una matriz de confusión
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```

Redes Neuronales Artificiales

```
# Redes Neuronales Artificiales -Clasificación
# Librerías
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline

# Cargar dataset
dataframe = pandas.read_csv("Celulares.csv", header=None)

dataset = pandas.read_csv('Celulares.csv')
X = dataset.iloc[:,:-1].values
Y = dataset.iloc[:, 20].values
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)

# MCreación del modelo
def baseline_model():
    model = Sequential()
    model.add(Dense(40, input_dim=20, activation='relu'))
    model.add(Dense(4, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

estimator = KerasClassifier(build_fn=baseline_model, epochs=10000,
batch_size=5, verbose=0)
kfold = KFold(n_splits=10, shuffle=True)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
= 0.2, random_state = 0);
estimator.fit(X_train, y_train)
```

```
y_pred  = estimator.predict(X_test)

# Predicción de nuestros modelos
y_pred  = estimator.predict(X_test)
from sklearn.metrics import r2_score
r2 =r2_score(y_test, y_pred)
```