Contenido

[1. Aspectos clave de machine learning. 3](#_Toc146020546)

[1.1 Aprendizaje supervisado. 3](#_Toc146020547)

[1.1.1 Problemas de clasificación. 4](#_Toc146020548)

[1.1.2 Problemas de regresión. 5](#_Toc146020549)

[1.2 Aprendizaje no supervisado. 5](#_Toc146020550)

[1.3 Aprendizaje por refuerzo. 5](#_Toc146020551)

[2. Flujos básicos de trabajo del machine learning. 5](#_Toc146020552)

[2.1 Representación: Ingeniería de características 5](#_Toc146020553)

[2.1.1 Pretratamiento de la información. 8](#_Toc146020554)

[2.1.2 Visualización de la información. 8](#_Toc146020555)

[2.2 Representación: Selección de modelo (Model Selection). 10](#_Toc146020556)

[2.3 Evaluación. 10](#_Toc146020557)

[2.4 Optimización. 10](#_Toc146020558)

[3. Librerías utilizadas en el machine-learning 11](#_Toc146020559)

[3.1 Scipy 11](#_Toc146020560)

[3.2 Scikit-learn 11](#_Toc146020561)

[3.2.1 Conjunto de datos incorporados (sklearn.datasets). 12](#_Toc146020562)

[3.2.2 Conjuntos sintéticos. 12](#_Toc146020563)

[3.3 Pandas 13](#_Toc146020564)

[3.4 MatPilob 14](#_Toc146020565)

[4. Conceptos transversales en los algoritmos de machine learning. 15](#_Toc146020566)

[4.1 Generalización. 15](#_Toc146020567)

[4.2 Sobreajuste (overfitting). 15](#_Toc146020568)

[4.2.1 Sobreajuste de clasificación 15](#_Toc146020569)

[4.2.2 Sobreajuste en regresión. 17](#_Toc146020570)

[4.3 Subajuste (underfitting) 17](#_Toc146020571)

[4.3.1 Subajuste en regresión. 18](#_Toc146020572)

[4.3.2 Subajuste en clasificación. 18](#_Toc146020573)

[4.4 Sesgo (Bias). 18](#_Toc146020574)

[4.5 Varianza (Variance). 19](#_Toc146020575)

[4.6 "Memory-based" o "instance-based learning" 19](#_Toc146020576)

[5. Algoritmos de machine learning y aplicaciones en python. 20](#_Toc146020577)

[5.1 Métodos comunes en machine learning. 20](#_Toc146020578)

[5.1.1 Preparación del conjunto de datos: separación en conjunto de entrenamiento y prueba con la función train\_test\_split [Scikit-learn] 20](#_Toc146020579)

[6. K vecinos más cercanos o K-NN. 21](#_Toc146020580)

[6.1 Descripción teórica. 21](#_Toc146020581)

[6.2 Parámetros importantes. 22](#_Toc146020582)

[6.3 K vecinos cercanos o K-NN en clasificación. 22](#_Toc146020583)

[6.3.1 Teoría: K-NN 22](#_Toc146020584)

[6.3.2 Aplicación en Python: K-NN 23](#_Toc146020585)

[6.3.3 Visualización con un gráfico de las áreas para distintos valores de k. 26](#_Toc146020586)

[6.3.4 Observaciones importantes y anécdotas aprendidas en proyectos reales. 26](#_Toc146020587)

[6.4 K vecinos cercanos o K-NN en regresión. 26](#_Toc146020588)

[7. Regresión Lineal. 27](#_Toc146020589)

[7.1 Mínimos cuadrados. 27](#_Toc146020590)

[7.1.1 Descripción teórica. 27](#_Toc146020591)

[7.1.2 Mínimos cuadrados en Python 29](#_Toc146020592)

[7.2 Regresión Ridge. 30](#_Toc146020593)

[7.3 Regresión Polinomial. 30](#_Toc146020594)

[7.4 Regresión Lógica. 30](#_Toc146020595)

[8. Análisis de algoritmos a nivel global. 31](#_Toc146020596)

[8.1 Tabla resumen ventajas/desventajas 31](#_Toc146020597)

[8.2 ¿Cuándo utilizar un algoritmo u otro? 32](#_Toc146020598)

[8.3 Contrastes entre algoritmos. 33](#_Toc146020599)

[8.3.1 Uso de regresión lineal con K-NN y método de mínimos cuadrados. 33](#_Toc146020600)

# Aspectos clave de machine learning.

El aprendizaje automático, o "machine learning" en inglés, es una rama de la inteligencia artificial que se centra en el diseño y construcción de algoritmos que permiten a las computadoras aprender a partir de y hacer predicciones o decisiones basadas en datos. En lugar de programar reglas específicas para realizar una tarea, en el aprendizaje automático, las máquinas se entrenan utilizando grandes conjuntos de datos y algoritmos que les dan la capacidad de aprender cómo realizar la tarea.

El machine Learning puede ser clasificado en tres grandes tipos:

* **Aprendizaje Supervisado:** Se entrena el modelo con datos etiquetados, es decir, con ejemplos donde ya conocemos la entrada y la salida esperada.
  + **Problemas de Clasificación:** La salida es discreta. Por ejemplo, determinar si un email es "spam" o "no spam".
  + **Problemas de Regresión:** La salida es un valor continuo. Ejemplo: predecir el precio de una casa basado en características como su tamaño y ubicación.
* **Aprendizaje No Supervisado:** Se trabaja con datos sin etiquetar, buscando identificar estructuras o patrones.
  + **Clustering:** Agrupar datos similares. Ejemplo: segmentar clientes en un mercado según sus hábitos de compra.
  + **Detección de Anomalías:** Identificar datos que no siguen un patrón usual. Por ejemplo, detectar transacciones fraudulentas en una tarjeta de crédito.
* **Aprendizaje por Refuerzo:** El modelo aprende a tomar decisiones basándose en recompensas o penalizaciones por sus acciones. Ejemplo: entrenar un agente para jugar y mejorar en un videojuego; el agente recibe recompensas por acciones correctas y penalizaciones por errores.

## Aprendizaje supervisado.

**Aprendizaje supervisado:** Aprende a predecir valores objetivos a partir de información o datos etiquetados. Buscamos predecir una variable de salida dada una variable de entrada. Dentro del aprendizaje supervisado podemos distinguir entre:

* **Problemas de clasificación:** Cuando el valor objetivo (la salida) puede tomar un conjunto discreto de posibilidades. Algunos ejemplos:
* **Clasificación binaria:** el clasificador tiene dos valores objetivos. Por ejemplo: Un cáncer es de tipo maligno o benigno, una transacción económica es un fraude o no es un fraude, un regalo entre empresarios es un soborno o no es un soborno. Normalmente una clase se representará con el 0 y otra con el 1.
* **Clasificación multiclase**: El clasificador tiene que elegir entre un conjunto discreto de posibilidades, por ejemplo: dada una imagen de un animal, clasificarlo como un determinado animal, es decir, un perro, un gato, pájaro, etc. Dado un texto como entrada determinar que idioma es (en el mundo hay 7139 idiomas) , vemos que a pesar un número relativamente grande, seguimos ante un problema de clasificación multiclase.
* **Clasificación multi-etiqueta:** Hay varios valores objetivos que pueden ser asignados a una muestra de entrada, por ejemplo, si nuestro clasificador aplica géneros a una determinada película, esta película puede tener 1 o más géneros, por ejemplo, {acción, comedia} o {aventura, drama, ciencia-ficción}.

### Problemas de clasificación.

Un conjunto de entrenamiento es un conjunto de datos utilizado en machine learning para enseñar a un modelo o algoritmo cómo realizar una tarea específica. Este conjunto de datos contiene ejemplos de entrada junto con las correspondientes salidas deseadas o etiquetas, y se utiliza para entrenar al modelo, permitiéndole aprender patrones y relaciones en los datos. El objetivo del entrenamiento es capacitar al modelo para que pueda hacer predicciones o tomar decisiones precisas en datos no vistos o nuevos. Un ejemplo es el que sigue:

Clasificador

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Asunto del Correo Electrónico** |  | **Etiqueta (Clase)** |
|  | Oferta exclusiva: ¡Gane dinero rápido! |  | Spam |
|  | Confirmación de reserva de vuelo |  | No Spam |
|  | ¡Felicidades! Ha ganado un premio de lotería |  | Spam |
|  | Actualización importante de su cuenta bancaria |  | No Spam |
|  | ¡Descuento del 50% en compras en línea esta semana! |  | Spam |
|  | Factura pendiente de pago |  | No Spam |
|  | ¡Gana un iPhone gratis! |  | Spam |

#### ¿De dónde vienen las etiquetas?

Las etiquetas de salida pueden ser implícitas o explicitas, tal que:

* **Las etiquetas explicitas:** Las etiquetas explícitas son información conocida y proporcionada de manera directa en el conjunto de entrenamiento. Estas etiquetas son las salidas o resultados deseados asociados a cada ejemplo de entrada en el conjunto de datos. En problemas de clasificación, las etiquetas explícitas indican a qué clase o categoría pertenece cada entrada.

Por ejemplo, en un conjunto de datos de detección de spam de correo electrónico, las etiquetas explícitas serían "Spam" o "No Spam" asociadas a cada correo electrónico.

* **Las etiquetas implícitas** se refieren a información que no se proporciona directamente en el conjunto de entrenamiento, pero se infiere o se asume a través de ciertos métodos. A menudo, las etiquetas implícitas se generan a partir de la estructura de los datos o de la relación entre las características de entrada.

Por ejemplo, si un usuario hace la consulta “libro de magia” y hace click en el resultado “Harry potter y la piedra Filosofal” entonces, la etiqueta implícita será “Harry potter y la piedra Filosofal” para la entrada “libro de magia”.

### Problemas de regresión.

La regresión es uno de los pilares fundamentales del aprendizaje supervisado en el campo del machine learning. A diferencia de la clasificación, donde el objetivo es predecir etiquetas discretas (como "gato" o "perro"), en los problemas de regresión, el objetivo es predecir un valor continuo. Estos valores continuos pueden abarcar desde precios de viviendas, temperaturas, ventas futuras, entre otros, y son números reales.

En términos simples, la regresión implica trazar una línea (o una superficie en dimensiones superiores) que mejor se ajuste a los puntos de datos disponibles. Esta línea, denominada línea de regresión, permite hacer predicciones sobre nuevos datos que no estaban en el conjunto de entrenamiento. Es decir, es buscar una función matemática que permita predecir valores para casos que no se conocen.

## Aprendizaje no supervisado.

## Aprendizaje por refuerzo.

# Flujos básicos de trabajo del machine learning.

REPRESENTACIÓN

EVALUACIÓN

OPTIMIZACIÓN

La **fase de representación** en el aprendizaje automático está fundamentada por dos pasos:

* **Elegir como representar la información adecuadamente (Feature Engineering):** es el proceso de convertir datos brutos en un formato estructurado que el modelo pueda entender. La calidad de esta representación es crucial ya que determina qué tan bien un algoritmo puede aprender de los datos.
* **Elegir el algoritmo de aprendizaje que se utilizará (Model selection):** Una vez que los datos están en un formato adecuado, necesitas seleccionar un algoritmo para modelar y aprender de esos datos. Diferentes algoritmos tienen diferentes capacidades y limitaciones, y la elección dependerá de la naturaleza del problema, la cantidad y calidad de los datos, y otros factores.

La **fase de evaluación** en el aprendizaje automático esta fundamentada por x pasos:

## Representación: Ingeniería de características

La representación de la información, a menudo llamada ingeniería de características (feature engineering), es una de las etapas más fundamentales y cruciales en cualquier proyecto de aprendizaje automático. Una representación adecuada de la información puede facilitar el proceso de aprendizaje de un algoritmo, mientras que una mala representación puede limitar seriamente la capacidad de un algoritmo para encontrar patrones útiles en los datos.

La representación de la información implica tomar datos brutos y convertirlos en un conjunto estructurado de características que describan de manera efectiva el problema que se está tratando de resolver. Veamos esto en detalle:

* **Selección de Características (Feature Selection):** No todas las características disponibles en un conjunto de datos pueden ser útiles para un problema específico. La selección de características implica elegir un subconjunto óptimo de características que contribuyen más significativamente al rendimiento del modelo.

**Ejemplo:** Si estás construyendo un modelo para predecir enfermedades cardíacas, la edad y la presión arterial podrían ser características relevantes, mientras que el color del coche del paciente probablemente no lo sea.

* **Creación de Características (Feature Creation)**: A veces, las características originales pueden ser transformadas o combinadas para crear nuevas características que ofrezcan más información al modelo.

**Ejemplo:** En un dataset sobre propiedades inmobiliarias, puedes tener las características "ancho del terreno" y "largo del terreno". Multiplicando estas dos podrías crear una nueva característica llamada "área del terreno".

* **Codificación (Encoding):** Muchos algoritmos requieren que las entradas sean numéricas. Por lo tanto, las características categóricas (como colores, tipos de empleo, países, etc.) deben codificarse en un formato numérico. Las técnicas comunes incluyen "one-hot encoding", "ordinal encoding" y "binary encoding", entre otros.
* **Normalización y Escalado:** Los algoritmos, especialmente aquellos basados en cálculos de distancia como k-NN o SVM, son sensibles a la escala de las características. Por tanto, es común escalar características para que tengan una media de 0 y una desviación estándar de 1 (normalización) o para que estén en un rango específico, como [0,1].
* **Manejo de Datos Faltantes**: Los conjuntos de datos a menudo tienen valores faltantes. La representación adecuada también implica decidir cómo se tratarán estos valores. Puedes eliminarlos, imputarlos con un valor específico (como la media o la mediana), o usar técnicas más avanzadas como la imputación basada en modelos.
* **Transformaciones:** Dependiendo de la naturaleza del problema y la relación entre las características y la variable objetivo, puedes aplicar transformaciones matemáticas a las características. Las transformaciones logarítmicas, cuadráticas y exponenciales son ejemplos comunes.
* **Reducción de Dimensionalidad:** En algunos casos, especialmente cuando se tiene un número muy grande de características, puede ser útil reducir la dimensionalidad de los datos. Técnicas como el Análisis de Componentes Principales (PCA) o el t-SNE se utilizan a menudo para este propósito.

Dada la importancia de la representación de la información, es común que los científicos de datos pasen una cantidad significativa de tiempo en esta etapa. La intuición, el conocimiento del dominio y la exploración exhaustiva de los datos son esenciales para crear una representación efectiva que facilite el proceso de aprendizaje automático.

Imagen que contiene Tabla

Descripción generada automáticamente

Features (Características)

Imagen que contiene Gráfico

Descripción generada automáticamente

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

### Pretratamiento de la información.

Antes de alimentar nuestro algoritmo de aprendizaje con un conjunto de datos, es esencial realizar diversas tareas de preparación. Estas incluyen:

* identificar y eliminar datos redundantes que podrían introducir ruido.
* determinar cómo gestionar los datos faltantes.
* estar atentos a problemas como medidas inapropiadas.
* asignación errónea de tipos de datos a columnas.
* columnas que encapsulan múltiples tipos de información y deberían dividirse.
* unidades de medida inconsistentes y desequilibrio en la representación de clases específicas.

Este paso asegura que el modelo se entrena con datos limpios y estructurados, maximizando su eficacia y precisión.

### Normalización de características.

La normalización de características es una técnica utilizada en el preprocesamiento de datos para cambiar los valores de las características numéricas en un conjunto de datos a una escala común. La normalización es esencial para algoritmos que son sensibles a magnitudes/varianzas en las características, como, por ejemplo, K-NN, redes neuronales, support vector machines (SVM). Sin normalización, las características que están en una escala mayor pueden dominar el resultado, aunque tal vez no sean necesariamente más importantes.

Para abordar este problema, a menudo es útil reescalar las características de manera que todas se encuentren dentro del mismo rango. Los métodos que veremos serán:

* **MinMax Scaling**; Reescala las características de manera que estén en un rango específico, típicamente entre 0 y 1.

#### MinMax Scaling

como su nombre indica, implica reescalar las características de forma que queden dentro de un rango mínimo y máximo especificado, típicamente entre 0 y 1. Lo hace utilizando la siguiente fórmula:

Donde:

* es el valor original.
* Xmin​ es el valor mínimo en la característica.
* Xmax​ es el valor máximo en la característica.
* Xnormalizado​ es el valor normalizado.

Esta formula se aplica para cada una de las características, un ejemplo visual de las implicaciones de esta transformación se refleja en las siguientes imágenes:

Gráfico, Gráfico de dispersión

Descripción generada automáticamenteGráfico, Gráfico de dispersión

Descripción generada automáticamente

En Python quedaría de la siguiente forma:

Texto

Descripción generada automáticamente

### Visualización de la información.

La visualización permite confirmar la calidad de la limpieza y preprocesamiento realizados y detectar posibles patrones, anomalías o características de interés en los datos. Utilizar herramientas de visualización, como [Matplotlib](#_MatPilob) nos permite:

* **Verificar el preprocesamiento:** Tras limpiar y transformar tus datos, es útil visualizarlos para asegurarte de que todo se ha procesado como esperabas.
* **Detectar anomalías o outliers:** Los gráficos, como los “box plots” o los “scatter plots”, pueden ayudarte a identificar puntos de datos que no encajan con el resto.
* **Entender distribuciones:** Histogramas y gráficos de densidad pueden mostrarte cómo están distribuidos tus datos, lo cual es fundamental antes de elegir un modelo. Estas distribuciones pueden formar “clústeres” o agrupaciones que nos permiten detectar visualmente ciertos patrones.
* **Identificar relaciones:** Gráficos como los de dispersión o los mapas de calor de correlación pueden mostrar cómo se relacionan las características entre sí.

#### Feature pair-plot

Un "feature pair plot", comúnmente conocido como "pair plot", es una herramienta de visualización que permite examinar rápidamente las relaciones por pares entre múltiples características (o variables) en un conjunto de datos. Es especialmente útil para explorar visualmente las relaciones y distribuciones en conjuntos de datos con múltiples variables.

El pair plot típicamente muestra una matriz de gráficos de dispersión para cada par de características, y a lo largo de la diagonal de esta matriz, puede mostrar un histograma o un gráfico de densidad para cada característica individual. La idea detrás de esto es ofrecer una visión rápida y comprensiva de cómo cada variable se relaciona con las otras en el conjunto de datos. Por ejemplo:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Interfaz de usuario gráfica, Gráfico, Aplicación

Descripción generada automáticamente

## Representación: Selección de modelo (Model Selection).

## Evaluación.

## Optimización.

# Librerías utilizadas en el machine-learning

## Scipy

SciPy es una biblioteca de código abierto para Python que se utiliza para tareas de matemáticas, ciencia e ingeniería. Es construida sobre la biblioteca NumPy y provee una gran cantidad de módulos y funciones que facilitan operaciones de álgebra lineal, optimización, integración, interpolación, transformadas de Fourier, procesamiento de señales, y estadísticas, entre otros.

A diferencia de NumPy, que proporciona objetos para matrices multidimensionales y funcionalidades básicas para operar con ellas, SciPy incorpora funcionalidades específicas de alto nivel para tareas científicas y técnicas. Es una herramienta esencial en el ecosistema de Python para la computación científica y se integra bien con otras bibliotecas populares como Matplotlib para visualización y Pandas para manipulación de datos.

## Scikit-learn

Scikit-learn es una biblioteca de código abierto para Python que ofrece herramientas simples y eficientes para análisis y modelado de datos en el ámbito del aprendizaje automático. Provee una amplia variedad de algoritmos supervisados y no supervisados, incluyendo regresiones, clasificaciones, reducción de dimensionalidad y clustering, entre otros.

Además, ofrece herramientas para la selección y evaluación de modelos, así como para la preparación de datos. Scikit-learn se integra bien con otras bibliotecas de Python, como NumPy y Pandas, y es conocida por su documentación clara y su comunidad activa, lo que la hace una opción popular tanto para principiantes como para expertos en el campo del aprendizaje automático.

Algunos enlaces de interés son:

|  |  |  |
| --- | --- | --- |
| **Concepto** | **Enlace** | **Descripción** |
| Página Principal | <https://scikit-learn.org/> | Sitio oficial de scikit-learn con enlaces a la documentación, tutoriales, y más. |
| User Guide | <https://scikit-learn.org/stable/user_guide.html> | Visión general de las principales secciones de la biblioteca y guía sobre cómo y cuándo usar diversos métodos. |
| API Documentation | <https://scikit-learn.org/stable/modules/classes.html> | Lista exhaustiva de todas las clases, funciones y módulos en scikit-learn con detalles. |
| Tutoriales | <https://scikit-learn.org/stable/tutorial/index.html> | Conjunto de tutoriales para iniciarse con scikit-learn. |
| Ejemplos | <https://scikit-learn.org/stable/auto_examples/index.html> | Colección de ejemplos prácticos que muestran aplicaciones reales de scikit-learn. |
| Developer Guide | <https://scikit-learn.org/stable/developers/index.html> | Guía para aquellos interesados en contribuir al desarrollo de scikit-learn. |
| Mailing List y Foros | <https://scikit-learn.org/stable/whats_new/v0.22.html#id35> | Puntos de contacto para discusiones y preguntas relacionadas con scikit-learn. |

Nota: Está librería usa SciPy (Science Python) y Numpy

### Conjunto de datos incorporados (sklearn.datasets).

scikit-learn (también conocido como sklearn) proporciona varios conjuntos de datos propios que se utilizan comúnmente para fines de aprendizaje automático y minería de datos. El módulo sklearn.datasets incluye una serie de conjuntos de datos incorporados que son útiles para la práctica y la experimentación con algoritmos de aprendizaje automático. Aquí hay algunos ejemplos de conjuntos de datos incorporados en scikit-learn:

* load\_iris: Contiene datos sobre flores iris y se utiliza para tareas de clasificación multiclase.

Calendario

Descripción generada automáticamente

* load\_digits: Contiene imágenes de dígitos escritos a mano y se utiliza para tareas de reconocimiento de dígitos.
* load\_wine: Contiene datos sobre vinos y se utiliza para tareas de clasificación multiclase. load\_diabetes: Contiene datos relacionados con la diabetes y se utiliza para tareas de regresión.
* load\_boston: Contiene datos sobre precios de viviendas en Boston y se utiliza para tareas de regresión.

Estos conjuntos de datos incorporados son útiles para aprender a utilizar las funciones de scikit-learn y para probar algoritmos de aprendizaje automático. Puedes cargar estos conjuntos de datos y comenzar a trabajar con ellos en tus propios proyectos de aprendizaje automático.

### Conjuntos sintéticos.

En el contexto del aprendizaje automático y la ciencia de datos, un "synthetic dataset" o conjunto de datos sintético se refiere a un conjunto de datos que no se obtiene a partir de eventos del mundo real, sino que se genera programáticamente. Estos conjuntos de datos se crean a menudo para simular ciertos tipos de comportamientos o patrones que se quieren estudiar sin tener que depender de datos reales, que pueden ser difíciles de obtener, incompletos o sesgados. La generación de conjuntos de datos sintéticos es particularmente útil en situaciones donde:

* Se quiere estudiar el comportamiento de un algoritmo bajo condiciones controladas.
* Se necesitan datos de ejemplo para demostraciones o tutoriales.
* Se quiere simular tipos específicos de estructuras de datos o patrones.
* Los datos reales son inaccesibles, limitados o presentan problemas de privacidad.

#### Make\_regression.

La función make\_regression es una herramienta proporcionada por scikit-learn, una de las bibliotecas más populares para aprendizaje automático en Python. Esta función se utiliza para generar conjuntos de datos sintéticos específicamente diseñados para problemas de regresión.

Los principales parámetros de make\_regression incluyen:

* **n\_samples:** Número de muestras (filas) en el conjunto de datos.
* **n\_features:** Número total de características (columnas).
* **n\_informative**: Número de características que son informativas, es decir, que realmente influyen en la salida.
* **noise:** Desviación estándar del ruido gaussiano añadido a la salida.
* **bias:** Bias o intercepto en la salida.
* **random\_state:** Semilla para garantizar la reproducibilidad de los datos generados.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Ilustración — Ejemplo código y gráfico asociado para visualizar una gráfica de dispersión

## Pandas

[Pandas](https://pandas.pydata.org/) es una biblioteca de código abierto en Python diseñada para la manipulación y análisis de datos. Proporciona estructuras de datos flexibles y de alto rendimiento, como el DataFrame, que facilita el manejo de datos estructurados.

Pandas ofrece herramientas robustas para importar/exportar datos desde distintos formatos (como CSV, Excel y SQL), filtrar y limpiar datos, realizar operaciones de agregación, combinar o unir conjuntos de datos, y visualizar información básica. Gracias a su versatilidad y eficiencia, Pandas se ha convertido en una herramienta esencial en ciencia de datos y análisis estadístico en Python.

## MatPilob

Matplotlib es una biblioteca de visualización en Python que proporciona herramientas flexibles para crear una variedad de gráficos y visualizaciones. Es una herramienta esencial para científicos e ingenieros que trabajan en el análisis de datos o la investigación científica.

# Conceptos transversales en los algoritmos de machine learning.

## Generalización.

La generalización se refiere a la habilidad de un modelo de aprendizaje automático para hacer predicciones precisas y relevantes sobre datos no vistos previamente, es decir, datos que no formaron parte del conjunto de entrenamiento. Un buen grado de generalización indica que el modelo ha capturado patrones inherentes y significativos de los datos con los que fue entrenado, en lugar de simplemente memorizar esos datos. Es resumen, la generalización es **aplicar el conocimiento aprendido durante el entrenamiento a nuevos datos no vistos**.

Subyacente a este proceso, el algoritmo presupone que los datos futuros, aunque no vistos, compartirán propiedades y características similares con el conjunto de entrenamiento. Sin embargo, esta suposición puede no ser siempre válida, llevando a problemas potenciales. Uno de los más prominentes es el "overfitting" o sobreajuste. El sobreajuste ocurre cuando un modelo se ajusta demasiado a las particularidades y ruidos de los datos de entrenamiento, sacrificando su capacidad de generalización.

## Sobreajuste (overfitting).

El sobreajuste es una condición en la que un modelo de machine learning se adapta de manera excesiva a los datos de entrenamiento, capturando el ruido y las fluctuaciones aleatorias en los datos en lugar de aprender la relación subyacente entre las variables. Como resultado, el modelo tiene un rendimiento excelente en los datos de entrenamiento, pero un rendimiento deficiente en datos no vistos, lo que se conoce como el conjunto de prueba o datos de validación. El sobreajuste ocurre cuando el modelo es demasiado complejo en relación con la cantidad y calidad de los datos de entrenamiento disponibles. Aquí hay algunas razones comunes que pueden llevar al sobreajuste:

* **Complejidad del modelo:** un modelo excesivamente complejo con muchas características, capas o parámetros puede ser propenso al sobreajuste, especialmente si la cantidad de datos de entrenamiento es limitada.
* **Cantidad limitada de datos:** El sobreajuste es más probable cuando tienes una cantidad pequeña de datos de entrenamiento. Cuantos menos ejemplos tengas para entrenar el modelo, mayor será el riesgo de que el modelo se ajuste en exceso.
* **Ruido en los datos:** Si tus datos de entrenamiento contienen ruido o errores aleatorios, el **modelo podría capturar esos errores y sobre-ajustarse a ellos** en lugar de aprender la verdadera relación entre las variables.
* **Características irrelevantes:** Incluir características irrelevantes o redundantes en el modelo puede conducir al sobreajuste, ya que el modelo puede intentar ajustarse a estas características sin valor predictivo real.
* **Interacciones no lineales:** En algunos casos, incluso modelos lineales pueden sobre ajustarse si intentan capturar relaciones no lineales en los datos de entrenamiento sin suficientes ejemplos para hacerlo con precisión.

### Sobreajuste de clasificación

[Gráfico, Gráfico de dispersión

Descripción generada automáticamente](https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76)En este caso, en la gráfica de la derecha tenemos un sobre ajuste del modelo para capturar detalles muy específicos, que pueden ser ruido o simplemente casos muy puntuales, haciéndonos perder generalización, en la gráfica de la izquierda podemos apreciar que el modelo es menos sensible a los casos puntuales como son las dos “X” que están entre los círculos, es decir, no se deja influenciar de manera sencilla por casos puntuales que no suelen ser los habituales. Además, los limites de decisión entre una clase u otra serán más variantes.

Otro ejemplo de sobreajuste en clasificación se puede ver con el algoritmo de [k vecinos más cercanos](#_K_vecinos_cercanos) para distintos valores de K, tal que:

Gráfico

Descripción generada automáticamente

En el algoritmo k-NN (k-Nearest Neighbors), cuando aumentamos el número k:

* **Aumenta la robustez frente al ruido**: Al considerar más vecinos para hacer una predicción, el algoritmo se vuelve menos sensible a puntos de datos ruidosos o atípicos. Un valor de k pequeño, como k=1, es más vulnerable al ruido en el dataset, y puede resultar en un modelo con alto sobreajuste (overfitting).
* **Borde de decisión más suave**: Un valor de k más grande tiende a suavizar el borde de decisión del modelo. Si imaginas la gráfica de los datos y el borde de decisión, con k=1, el borde podría ser muy errático, siguiendo de cerca los puntos de entrenamiento individuales. A medida que kk aumenta, el borde se vuelve más suave y menos errático.
* **Mayor riesgo de subajuste:** Si bien un k muy pequeño puede conducir al sobreajuste, un k muy grande puede hacer que el algoritmo sea insensible a las diferencias más sutiles en las distribuciones de clases y conducir a un subajuste (underfitting). Si k es igual al número total de puntos de datos, entonces el modelo simplemente predecirá la clase dominante en todo momento.

### Sobreajuste en regresión.

Un ejemplo de sobreajuste en una regresión puede ser el siguiente:

Imagina que estás tratando de entrenar un modelo de machine learning para predecir el precio de las casas en función de su tamaño en metros cuadrados, y tienes un conjunto de datos de entrenamiento relativamente pequeño, digamos solo 10 ejemplos de casas con sus tamaños y precios. Ahora, intentas usar un **modelo extremadamente complejo**, como una red neuronal profunda con muchas capas y neuronas.

Aquí es donde surge el problema del sobreajuste. Dado que tienes solo 10 ejemplos, el modelo puede aprender a ajustarse perfectamente a esos datos individuales, incluso a pequeños detalles y ruidos en los datos. En otras palabras, puede aprender a "memorizar" los precios específicos de esas 10 casas en lugar de aprender la relación general entre el tamaño y el precio de las casas.

El modelo sobre ajustado será muy bueno para predecir los precios de las 10 casas en el conjunto de entrenamiento, pero su desempeño será terrible cuando se le presenten nuevas casas que no estaban en ese conjunto. Esto se debe a que el modelo se ha centrado demasiado en los detalles específicos de los datos de entrenamiento y **no ha logrado capturar patrones generales que se apliquen a todas las casas**.

[Gráfico, Gráfico de dispersión

Descripción generada automáticamente](https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76)

Ilustración — Visualización gráfica de sobreajuste

Tal y como se muestra en la anterior imagen, tenemos que el modelo sobre ajustado, no será capaz de adaptarse a puntos nuevos porque esta muy ajustado a el conjunto de datos del entrenamiento. El modelo de la grafica “overfitted” es mucho más complejo, detecta patrones y casos específicos con muy buena precisión, pero solo para el conjunto de datos de entrenamiento.

## Subajuste (underfitting)

El subajuste es una condición en la que un modelo de machine learning es demasiado simple para capturar la estructura real y los patrones en los datos de entrenamiento. Como resultado, el modelo tiene un rendimiento deficiente tanto en los datos de entrenamiento como en los datos de prueba y no puede hacer predicciones precisas.

* **Modelo muy simple:** Utilizar un modelo que carece de capacidad para representar la complejidad de los datos es una causa principal de sub-ajuste.
* **Falta de características (features) relevantes:** Si el conjunto de características utilizado para entrenar el modelo no incluye variables importantes o características que capturan la relación subyacente, el modelo puede sub-ajustarse.
* **Cantidad insuficiente de datos:** Cuando hay muy pocos ejemplos de entrenamiento para que el modelo aprenda patrones significativos, es probable que ocurra el sub-ajuste.
* **Regularización excesiva:** La aplicación de una regularización excesiva (por ejemplo, L1 o L2) puede hacer que el modelo sea demasiado simplista y cause subajuste.

### Subajuste en regresión.

Imagina que estás entrenando un modelo de regresión lineal para predecir la temperatura diaria en función de la fecha en un conjunto de datos históricos. Sin embargo, decides utilizar un modelo lineal simple con una sola característica, como el día del año. A medida que ajustas el modelo a los datos de entrenamiento, notas que las predicciones del modelo son muy inexactas tanto en los datos de entrenamiento como en los datos de prueba. El modelo subajusta los datos porque es demasiado simple para capturar las complejas variaciones de temperatura a lo largo del año, y falta información importante, como las estaciones y las tendencias a largo plazo, que podrían haberse incluido mediante características adicionales.

[Gráfico, Gráfico de dispersión

Descripción generada automáticamente](https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76)

<https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76>

Como apreciamos, en el caso de subajuste (underfitting) vemos que no se captura la complejidad de la distribución de datos, sino que lo simplifica demasiado teniendo malos resultados tanto para el conjunto de entrenamiento como para el conjunto de nuevos datos.

### Subajuste en clasificación.

## Regularización.

La regularización es una técnica utilizada en machine learning y estadísticas para prevenir el sobreajuste de los modelos. El sobreajuste ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y, como resultado, tiene un rendimiento pobre en datos no vistos o de prueba.

En esencia, la regularización añade una penalización al modelo para restringir su complejidad. Al hacerlo, se busca conseguir un equilibrio entre el ajuste a los datos de entrenamiento y la simplicidad del modelo.

## Sesgo (Bias) y Varianza.

El sesgo se refiere a la simplificación excesiva de un modelo, lo que hace que no se ajuste bien a los datos de entrenamiento y, por lo tanto, no pueda capturar patrones complejos en los datos. Un modelo con alto sesgo tiende a subestimar o sobreestimar sistemáticamente las predicciones.

El sesgo de un algoritmo implica simplificar o generalizar las predicciones, lo que puede llevar a perder los detalles o casos particulares en los datos. Esto se hace para que el modelo sea más simple y pueda realizar predicciones sobre datos nuevos de manera más general, pero a menudo a expensas de la precisión en casos específicos.

Un modelo con **alto sesgo** tiende a no adaptarse bien a los datos de entrenamiento y, por lo tanto, puede no adaptarse bien a datos nuevos. Esto se debe a que un alto sesgo implica que el modelo está haciendo suposiciones simplificadas que no se ajustan adecuadamente a la estructura real de los datos.

Un modelo con **bajo sesgo** tiene más flexibilidad y puede adaptarse mejor a los datos de entrenamiento. Sin embargo, si se vuelve demasiado flexible (alta varianza), corre el riesgo de sobreajustarse a los datos de entrenamiento y no generalizar bien a datos nuevos.

La **varianza** se refiere a la sensibilidad excesiva de un modelo a pequeñas fluctuaciones o ruido en los datos de entrenamiento. Un modelo con alta varianza se ajusta demasiado a los datos de entrenamiento y, como resultado, puede tener un rendimiento pobre en datos que no ha visto antes (datos de prueba).

## Multicolinealidad

Es un concepto transversal en estadísticas y machine learning que surge cuando dos o más características predictoras en un modelo están altamente correlacionadas, lo que significa que una puede predecirse a partir de las otras con un alto grado de precisión.

## "Memory-based" o "instance-based learning"

**"Memory-based" o "instance-based learning"** se refiere a un tipo de aprendizaje en el que el modelo "recuerda" ejemplos de la base de datos de entrenamiento en lugar de aprender explícitamente una representación o una función general a partir de esos ejemplos.

# Algoritmos de machine learning y aplicaciones en python.

## Métodos comunes en machine learning.

### Preparación del conjunto de datos: separación en conjunto de entrenamiento y prueba con la función train\_test\_split [Scikit-learn]

Para construir y validar modelos de aprendizaje automático de manera eficiente, es esencial dividir nuestros datos en dos segmentos distintos: el conjunto de entrenamiento y el conjunto de prueba. El conjunto de entrenamiento se utiliza para "enseñar" o entrenar al modelo, mientras que el conjunto de prueba sirve para evaluar su rendimiento en datos no vistos previamente, simulando cómo el modelo funcionaría en situaciones reales.

La función **train\_test\_split** de la biblioteca Scikit-learn facilita este proceso. Con un simple llamado, podemos dividir nuestro conjunto de datos — que incluye tanto las características como las etiquetas asociadas — en estos dos conjuntos esenciales. Esta división no solo garantiza que el modelo no se sobreajuste a los datos de entrenamiento, sino que también proporciona una evaluación honesta de su capacidad para generalizar a nuevos datos. Está función tiene la forma que sigue:

Imagen que contiene Tabla

Descripción generada automáticamente

#### Train\_size y Test\_size

Aquí es importante entender que a no ser que se especifique con el parámetro opcional “test\_size” o “train\_size”, se asignara un 25% del conjunto de datos para pruebas y otro 75% para entrenamiento. Por ejemplo:

* Si tienes 1000 muestras en total y no especificas test\_size, obtendrás 750 muestras en el conjunto de entrenamiento y 250 muestras en el conjunto de prueba.

Sin embargo, si especificamos test\_size en tanto por 1, entonces especificaremos cuanto porcentaje del conjunto total será dedicado para test y así comprobar la efectividad del clasificador.

#### Random\_state

El parámetro **random\_state** determina la semilla utilizada por el generador de números aleatorios al seleccionar muestras para los conjuntos de entrenamiento y prueba. Establecer un valor fijo para random\_state asegura que, cada vez que se ejecute train\_test\_split con los mismos datos y parámetros, se obtengan las mismas divisiones de entrenamiento y prueba, lo que garantiza la reproducibilidad en la división de datos. Sin un random\_state fijo, las muestras seleccionadas para cada conjunto pueden variar con cada ejecución, debido a la naturaleza aleatoria del proceso.

# K vecinos más cercanos o K-NN (Clasificación y regresión).

## Introducción.

k-NN significa "k-Nearest Neighbors" (k-Vecinos Más Cercanos) y es uno de los algoritmos más simples y fundamentales en el aprendizaje supervisado. Es un algoritmo no paramétrico utilizado tanto **para clasificación como para regresión.**

El algoritmo **k-Nearest Neighbors (k-NN)** es un ejemplo clásico de aprendizaje basado en instancias o memoria. No construye explícitamente un modelo general durante la fase de entrenamiento; en su lugar, simplemente "recuerda" los ejemplos de entrenamiento y hace predicciones basándose en la similitud de las nuevas instancias con las instancias almacenadas. La k hace referencia al número de vecinos que el algoritmo utilizará para hacer su predicción

Dado un conjunto de entrenamiento con sus etiquetas asociadas (clases) o valores objetivos (números reales) y dada una nueva muestra para ser clasificada, el algoritmo de K-NN sigue los siguientes pasos:

Primero busca los k vecinos más cercanos a esa muestra

Obtiene las etiquetas(clases) o valores objetivos(regresión) de los “k” vecinos más cercanos.

Predice la etiqueta para la nueva muestra:

En clasificación: eligiendo por mayoría simple la etiqueta que más se repita en los vecinos.

En regresión: normalmente calculando la media

**1. Métrica de distancia:**

* Es crucial seleccionar una métrica de distancia adecuada para medir cuán "cercanas" están entre sí dos instancias.
* La elección de la métrica dependerá en gran medida de la naturaleza de tus datos. Por ejemplo:
  + **Distancia euclidiana**: Adecuada para datos numéricos.
  + **Distancia de Manhattan**: También para datos numéricos, pero mide la distancia en trayectorias rectangulares.
  + **Distancia de Hamming**: Adecuada para datos categóricos.

**2. Determinación del valor de 'k':**

* El valor de 'k' decide cuántos vecinos más cercanos se considerarán al hacer una predicción.
* Es esencial elegir un 'k' óptimo para asegurar un equilibrio entre evitar el ruido (valores atípicos) y no ser demasiado general.
* Una elección común es seleccionar un 'k' impar para evitar posibles empates.

**3. Ponderación opcional de los vecinos:**

* En lugar de tratar a todos los vecinos con igual importancia, es posible asignar pesos a los vecinos en función de su distancia al punto de consulta.
* Los vecinos que están más cerca pueden tener más influencia en la decisión que aquellos que están más lejos.
* Una técnica común es ponderar las contribuciones de los vecinos en función del inverso de su distancia al punto de consulta.

**4. Método de agregación:**

* Una vez que hayas identificado los 'k' vecinos más cercanos, necesitas una estrategia para combinar su información y hacer una predicción.
* **Para clasificación**: Una estrategia común es tomar una "votación por mayoría", en la que la clase que aparece con mayor frecuencia entre los 'k' vecinos es la clase predicha para el punto de consulta.
* **Para regresión**: Una estrategia común es calcular el promedio de las salidas (valores) de los 'k' vecinos.

## Parámetros importantes.

Complejidad del modelo:

* Numero de vecinos: el valor por defecto suele ser 5
* Métrica: Función que determina distancia entre los puntos de datos: por defecto suele ser distancia Minkowski con p = 2, es decir, la distancia euclidiana.

## K vecinos cercanos o K-NN en clasificación.

### Particularidades en clasificación.

Gráfico

Descripción generada automáticamente

En este grafico se puede apreciar donde se clasificaría una nueva muestra según el ancho y la altura de una fruta cuando k=1, es decir, siempre se clasificará una nueva muestra con la clase que tenga el punto más cercano de los ya existentes en el conjunto.

### Aplicación en Python: K-NN

Una vez importado un conjunto de datos como un dataframe de pandas, tal y como se ve en el apartado de [train\_test\_split](#_Preparación_del_conjunto) debemos separar dos conjuntos, uno para entrenar nuestro algoritmo de aprendizaje y otro para testearlo y así ver su precisión. Tenemos como ejemplo el siguiente conjunto de datos:

Tabla

Descripción generada automáticamente

La manera de aplicar el algoritmo K-NN se explica en las siguientes imágenes:

Texto

Descripción generada automáticamente

A partir de aquí, podemos generar nuevas predicciones con datos nuevos que la instancia del algoritmo K-nn no haya visto antes, por ejemplo:

Texto

Descripción generada automáticamente

### Visualización con un gráfico de las áreas para distintos valores de k.

K=5

Gráfico

Descripción generada automáticamente

### Observaciones importantes y anécdotas aprendidas en proyectos reales.

* Knn.score nos dará en tanto por 1 el número de acierto con el conjunto de testeo.
* Cuando K=1 la predicción es sensible a el ruido, datos que salen del comportamiento habitual, datos sin etiqueta y es más probables que se produzca sobreajuste, es decir, el modelo predice bien datos del propio conjunto de entrenamiento, pero no predice bien nuevas muestras.
* Una manera de obtener un valor optimo de K es hacer diferentes separaciones del conjunto de datos inicial en subconjuntos de tipo entrenamiento y tipo test, y buscar aquel donde la función knn.score nos devuelva el mayor score.
* Cuando K es mayor, es más robusto ante el ruido de otros puntos, pero resulta en límites de clasificación menos detallados.
* Cuando los dat

## K vecinos cercanos o K-NN en regresión.

Cuando k-NN se utiliza para regresión, el objetivo es predecir un valor continuo. La predicción se realiza de la siguiente manera:

1. Dado un nuevo punto de entrada para el cual deseamos hacer una predicción, encontramos los k puntos del conjunto de entrenamiento que están más cerca de ese punto.
2. Para estos k vecinos, calculamos la media (o a veces la mediana) de sus valores de salida.
3. Esta media se utiliza como la predicción del valor de salida para el nuevo punto de entrada.

Por ejemplo, si usamos k-NN en un problema de regresión para predecir precios de casas y para un determinado conjunto de características de una casa, los 3 vecinos más cercanos tienen precios de $100k, $110k y $105k, el algoritmo k-NN (con k=3) predeciría un precio de $(100 + 110 + 105) / 3 = $105k.

[Gráfico, Gráfico de dispersión

Descripción generada automáticamente](https://www.coursera.org/learn/python-machine-learning/lecture/I1cfu/k-nearest-neighbors-classification-and-regression)

Ilustración — Machine Learning Aplicado a Python - K-NN en regresión

En la anterior imagen podemos apreciar 3 graficos, en el primero se muestran los puntos de datos originales, en el segundo caso, se muestran con triángulos (para poder diferenciarlos de los círculos originales que representan los datos de entrenamiento), estos triángulos representan la predicción del algoritmo para cualquier valor de “Input feature”.

El coeficiente de determinación, conocido como o "r-squared", se utiliza para evaluar la calidad de las predicciones en un modelo de regresión. Este coeficiente mide la proporción de la variabilidad total de la variable dependiente que es explicada por el modelo. El valor de varía entre 0 y 1: un de 0 indica que el modelo no explica ninguna de la variabilidad de los datos de respuesta alrededor de su media, lo que implica una predicción inadecuada; mientras que un de 1 indica que el modelo explica toda la variabilidad, representando una predicción perfecta. Es esencial considerar que, aunque un valor más alto de sugiere un mejor ajuste del modelo, no garantiza que el modelo sea apropiado ni que las predicciones sean precisas en todos los contextos.

# Regresión Lineal (regresión).

## Introducción.

Un problema de regresión lineal expresa el valor de salida objetivo en términos de la suma ponderada de las variables de entrada, es decir, tiene la forma de una ecuación lineal de “n” variables independientes, donde cada variable tiene un coeficiente asociado y opcionalmente hay un término independiente. El objetivo es encontrar los coeficientes que minimizan la suma de los errores cuadrados entre las predicciones del modelo y los valores observados.

El **modelo de regresión lineal simple** (con una sola variable independiente) tiene la siguiente forma:

Donde:

* es la variable dependiente (objetivo).
* ​ es el término independiente o intersección, en el contexto de machine learning es denominado Bias o intersección del modelo.
* ​ es el coeficiente de la variable ​ (la pendiente).

Para **regresión lineal múltiple** (con más de una variable independiente), la ecuación se expande de la siguiente manera:

Donde ​ son las n variables independientes y ​ son los coeficientes respectivos, y es el sesgo, termino independiente o intersección del modelo.

## Mínimos cuadrados.

### Introducción.

El método de mínimos cuadrados es una técnica estándar para estimar estos coeficientes β. Básicamente, **busca minimizar la suma de los cuadrados** de las diferencias entre los valores observados y los valores predichos por el modelo.

[Gráfico, Gráfico de dispersión

Descripción generada automáticamente](https://medium.com/analytics-vidhya/ordinary-least-square-ols-method-for-linear-regression-ef8ca10aadfc)

Valor predicho  
 Valor real observado : Error

Al minimizar la suma de los cuadrados estamos minimizando el error que se produce entre predicción y observación. Para minimizar tenemos que encontrar los valores óptimos de los coeficientes incluido el termino independiente, esto se hace a partir de los datos de entrenamiento.

### Mínimos cuadrados en Python.

A continuación, se muestra un fragmento de código que aplica el algoritmo de mínimos cuadrados:

Texto, Carta

Descripción generada automáticamente

Teniendo como resultado:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

## Regresión Ridge (Penalización L2).

Antes de entrar en detalle acerca de la regresión de Ridge, es importante recordar que un coeficiente representa la relación entre una característica y la variable objetivo. Un coeficiente grande sugiere que pequeños cambios en esa característica pueden llevar a grandes cambios en la predicción. Si este coeficiente es excesivamente grande, puede indicar que el modelo se está [sobre ajustando](#_Sobreajuste_(overfitting).) a los datos de entrenamiento o que está siendo afectado por [multicolinealidad](#_Multicolinealidad).

La **multicolinealidad es la interdependencia** entre variables independientes. Para entender por qué esto puede ser un problema, imaginemos un estudio en el que tratamos de determinar los factores que influyen en el precio de una casa. Si consideramos tanto el tamaño de la casa (en metros cuadrados) como el número de habitaciones como características, podríamos encontrarnos con un alto grado de multicolinealidad. Esto es porque, en general, las casas más grandes suelen tener más habitaciones. Entonces, cuando intentamos determinar el impacto individual de cada característica en el precio, nos resulta difícil debido a su alta correlación.

La penalización en Ridge es una forma de añadir sesgo al modelo. Recordemos que el sesgo se refiere a cuánto se alejan las predicciones del modelo de los datos reales debido a supuestos erróneos en el modelo. Al limitar la magnitud de los coeficientes con Ridge, introducimos un poco de sesgo para reducir la varianza y mejorar la capacidad de generalización del modelo.La regresión Ridge es una extensión de la regresión lineal que incorpora una penalización para los coeficientes del modelo. Mientras que la regresión lineal convencional utiliza el método de mínimos cuadrados para minimizar el error residual, la regresión Ridge añade un término de penalización que actúa sobre la magnitud de los coeficientes, es decir que penaliza los coeficientes grandes, haciendo una **regularización** que consigue con la siguiente formula:

Teniendo en cuenta que el objetivo es minimizar la función RSS entonces si se tienen “n” soluciones con igual valor mínimo, y ese mínimo es el más optimo, de esas “n” soluciones, teniendo en cuenta el nuevo criterio de penalización se elegirá aquella solución que minimiza el tamaño de los coeficientes.

Además, el parámetro α controla la intensidad de esta penalización: un α más grande implica una penalización más fuerte, y un α de 0 reduciría la regresión Ridge a la regresión lineal ordinaria. El valor por defecto de es 1.5

En esencia, la regresión Ridge busca **encontrar un equilibrio entre la adaptación fiel a los datos y la simplicidad del modelo**, mejorando así su capacidad de generalización. Además, cabe señalar que esta regresión tiene un doble objetivo:

* Ajustarse bien a los datos (minimizando el error residual)
* Mantener los coeficientes relativamente pequeños (minimizando la penalización).

La penalización evita que los coeficientes alcancen valores excesivamente grandes, lo cual puede ser un signo de sobreajuste o de inestabilidad debido a problemas como la multicolinealidad (que unas características o variables independientes dependan de otras)

Interfaz de usuario gráfica, Gráfico, Gráfico de líneas

Descripción generada automáticamente

Una de las características más interesantes que se aprecian en la gráfica es que los coeficientes nunca llegan a ser cero, por lo cual, no se eliminan características

## Regresion Lasso (Penalización L1).

La regresión Lasso (Least Absolute Shrinkage and Selection Operator) es otra variante de la regresión lineal que incorpora regularización, similar a la regresión Ridge. Sin embargo, hay una diferencia clave en la forma en que Lasso aplica la penalización.

Mientras que la regresión Ridge penaliza la suma de los cuadrados de los coeficientes (regularización L2), Lasso penaliza la suma absoluta de los coeficientes (regularización L1). La función objetivo en Lasso se define como:

Esta característica de penalización L1 tiene una consecuencia especial: Lasso tiene la capacidad de realizar selección de características. Esto significa que, durante el proceso de regularización, **algunos coeficientes pueden ser exactamente reducidos a cero**, resultando en un modelo más esparcido o más "simple". Por lo tanto, Lasso no solo evita el sobreajuste y maneja la multicolinealidad, sino que también puede **indicar cuáles características son las más relevantes para la predicción**.

En Python la regresión Lasso se implementa de la siguiente manera:

Texto

Descripción generada automáticamente

En el siguiente gráfico se muestra como evoluciona el valor de los coeficientes a medida que aumentamos Alpha (parámetro de regularización), en este caso, al contrario que en el caso de L2 (Ridge) podemos apreciar que los coeficientes que tienen asociada una característica menos relevante se van convirtiendo en cero a medida que aumenta el parámetro de regularización.

Interfaz de usuario gráfica, Gráfico, Gráfico de líneas

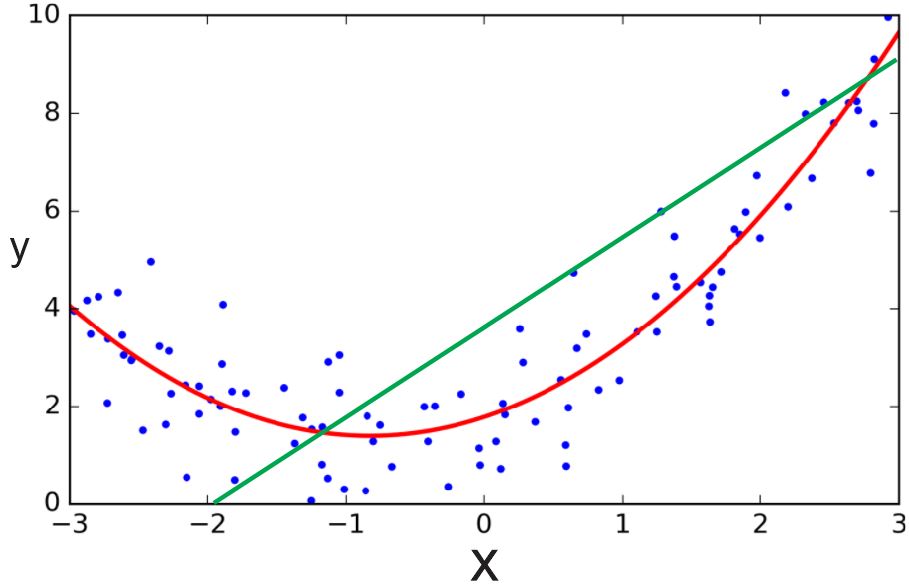
Descripción generada automáticamente

## Regresión Polinomial.

### Introducción.

La regresión polinomial es una forma de regresión lineal en la que la relación entre la variable independiente x y la variable dependiente y se modela como un polinomio de grado n. Es decir, en lugar de modelar la relación como , se podría modelar como , que es una ecuación polinómica de segundo grado.

Para lograr esto se deben agregar potencias de las características originales como nuevas características para modelar relaciones no lineales entre las características y la respuesta.

[](https://ardianumam.wordpress.com/2017/09/21/polynomial-regression-using-least-square-estimation/)

La función representada con la línea verde representa una regresión lineal utilizando el método de mínimos cuadrados y limitándose a utilizar una ecuación lineal de primer grado, mientras que la línea roja representa el mismo caso, pero no limitándose a una ecuación de primer grado, sino que se puede usar una función polinómica para captura patrones más complejos en la nube de puntos de la gráfica, teniendo un score más optimo.

Un efecto lateral de añadir más características es que un modelo más complejo tiene más probabilidades de que derive en un sobreajuste. Por ello, normalmente en la práctica, la regresión polinomial es hecha con algún tipo de regresión regularizada como la regresión Ridge.

### Regresión polinomial en Python.

En primer lugar, mostramos el caso de regresión lineal no polinomial junto con los datos que da como resultado:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

A continuación, aplicamos el mismo caso, pero con regresión polinomial, tal que:

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Como podemos apreciar, ha crecido considerablemente los valores de score, dando a entender que este modelo es mejor prediciendo datos. Finalmente, si normalizamos los datos antes de aplicar el modelo polinómico tendremos que:

Texto

Descripción generada automáticamente

Que produce el siguiente resultado:

Texto

Descripción generada automáticamente

En este caso tenemos un menos sobreajuste al conjunto de datos de entrenamiento.

# Regresión Lógica (clasificación).

## Introducción.

La regresión logística es un tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica basada en una o más variables predictoras. La variable de respuesta suele ser dicotómica, es decir, tiene dos posibles resultados (como sí/no, 1/0, verdadero/falso).

Mientras que en la regresión lineal se busca predecir una variable continua, en la regresión logística se modela la probabilidad de que una observación pertenezca a una de las categorías.

La regresión logística no se limita únicamente a la clasificación binaria. Aunque la forma más simple y comúnmente discutida de regresión logística es binaria, hay extensiones de este método para abordar problemas con más de dos categorías. Dos de estas extensiones son:

* **Regresión Logística Multinomial**: Esta es adecuada cuando la variable de respuesta tiene tres o más categorías que no tienen un orden particular. Por ejemplo, predecir el tipo de fruta (manzana, plátano, cereza) basado en ciertas características. En este modelo, se estima la probabilidad de que una observación pertenezca a cada categoría.
* **Regresión Logística Ordinal**: Si la variable de respuesta tiene un orden natural (como bajo, medio, alto), se puede usar la regresión logística ordinal. Aquí, las categorías son ordenadas, y el modelo estima la probabilidad de que una observación pertenezca a una categoría o a alguna de menor orden

## Regresión logística para clasificación binaria.

La formula de la regresión logística viene dada por la siguiente expresión:

La función logística fuerza a que el resultado final de sea un valor en 0 y 1, de tal forma de que para todos aquellos valores de que hagan que la función logística de como resultado un valor menor que 0.5 sean clasificados en una clase y para el caso de que sean mayores que 0.5 sean clasificados en la clase opuesta.

Gráficamente hablando la regresión logística para clasificación binaria se vería un plano (o una figura más compleja para problemas con dimensiones mayores) que separaría la nube de datos del conjunto de entrenamiento en dos espacios, uno para una clase y otro para la otra clase, visualmente sería como las siguientes graficas:

[Gráfico, Gráfico de dispersión

Descripción generada automáticamente](https://machine-learning.paperspace.com/wiki/logistic-regression)

Ilustración —Regresión logísitca de una dimensión (una entrada y una salida)

[Gráfico, Gráfico de dispersión

Descripción generada automáticamente](https://x-datainitiative.github.io/tick/auto_examples/plot_2d_linear_regression.html)

Ilustración — Regresión logistica de 2 dimensiones (2 entradas y una salida)

## Regresión logística: Regularización.

Una vez visualizadas las gráficas, es evidente cómo la regresión logística actúa como un "separador" de los datos en función de las clases. Pero, ¿qué pasa si los datos son muy complejos o si hay demasiadas características? Podríamos terminar con un modelo que se ajuste demasiado bien a los datos de entrenamiento, pero que no generalice bien a nuevos datos. Aquí es donde entra la regularización de Lasso (L1) o de Ridge (L2).

En Python, en particular con la biblioteca Scikit-Learn, cuando usas el LogisticRegression para clasificación, por defecto se aplica la regularización L2 (Ridge). El parámetro C en Scikit-Learn es **el inverso del parámetro** de regularización α que se vieron en los epígrafes de [Regresión Ridge](#_Regresión_Ridge_(Penalización) y [Regresión Lasso](#_Regresion_Lasso_(Penalización); es decir, **valores más pequeños de C** especifican una mayor regularización.

Además, es importante [normalizar la información](#_Normalización_de_características.) antes de aplicar el algoritmo de regresión logística, es decir, utilizar alguna técnica para que toda la información tenga la misma escala. A continuación se muestra un ejemplo de captura de código de Python en el que se entrena mediante regresión logística.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ilustración — Captura código de python

Dando como resultado:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

# Análisis de algoritmos a nivel global.

## Tabla resumen ventajas/desventajas

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nombre de algoritmo | Descripción breve | Ventajas | Desventajas | ¿Cuándo se debe descartar? | ¿Cuándo se debe elegir? |
| K-NN | Encuentra el k-ésimo vecino más cercano de un punto de datos y predice su valor objetivo en función de los valores objetivos de los vecinos. | Simple, fácil de implementar, robusto a los datos faltantes. | Puede ser sensible al ruido en los datos, puede ser menos preciso que otros algoritmos de regresión. | Datos con ruido, datos no lineales.  Cuando existen muchas características o muchas muestra se descarta | Datos con poca cantidad de ruido, datos lineales. |
| Regresión lineal | Encuentra una línea que mejor se ajuste a los datos. | Simple, fácil de interpretar, escalable.  Funciona bien cuando hay muchas características. | Puede no ser adecuada para datos no lineales, puede ser sensible a los outliers. | Datos no lineales, datos con outliers. | Datos lineales, datos con pocos outliers.  Optimo cuando hay muchas características. |
| Regresión logística | Encuentra una función logística que mejor se ajuste a los datos. | Adecuada para datos binarios, fácil de interpretar.  Es bueno para generalizar cuando hay muchas características. | Puede no ser adecuada para datos no lineales. | Datos no lineales. | Datos binarios, datos lineales. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Árbol de decisión | Divide los datos en subconjuntos cada vez más pequeños hasta que todos los datos estén en un solo subconjunto. | Fácil de entender e interpretar, robusto a los datos faltantes. | Puede no ser adecuado para datos no lineales, puede ser sensible a los outliers. | Datos no lineales, datos con outliers. | Datos lineales, datos con pocos outliers. |
| Random Forest | Combina múltiples árboles de decisión para mejorar la precisión. | Preciso, robusto a los datos faltantes, no sensible a los outliers. | Puede ser complejo de interpretar. | Datos con ruido, datos no lineales. | Datos con pocos outliers, datos lineales o no lineales. |
| Redes neuronales | Son un modelo de aprendizaje automático que se inspira en el cerebro humano. | Pueden aprender patrones complejos en los datos, son muy precisos. | Pueden ser difíciles de entrenar, requieren grandes cantidades de datos. | Datos con patrones complejos, datos con ruido. | Datos con patrones simples, datos sin ruido. |

## ¿Cuándo utilizar un algoritmo u otro?

## Contrastes entre algoritmos.

### Uso de regresión lineal con K-NN y método de mínimos cuadrados.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

La

Descartamos K-NN cuando

## Invocación de algoritmos inteligentes con scikit-learn