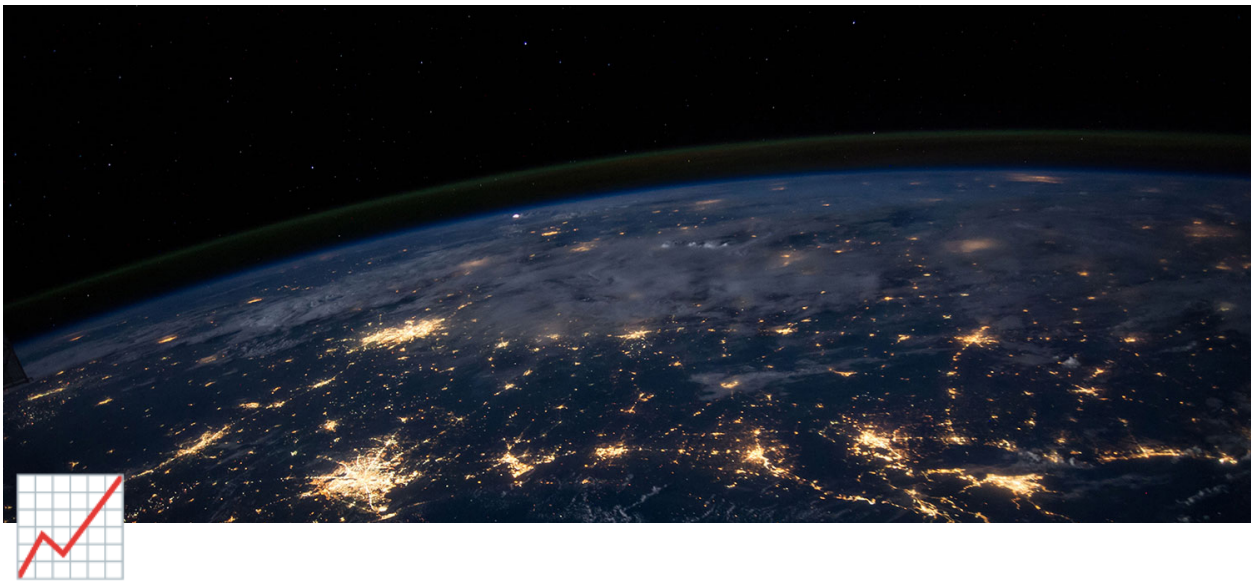


Modelos de Regresión

Ya vimos anteriormente que una forma de lograr encontrar una relación matemática entre una o varias *variables independiente* y continuas con una *variable dependiente* continua es a través del método **OLS** (gauss), o bien usando sus sucesores **Lasso (L1)** o **Ridge (L2)**.

También vimos que para lograr un **buen modelo** se puede hacer por dos caminos:

1. Criterios directos: separando el conjunto (set) de datos en: *entrenamiento, validación y testeo* de forma estructurada o bien usando el método de **K-Folds**.
2. Criterios indirectos: estos se utilizan *después* de entrenado el modelo, para evaluar su performance comparándolo con otros modelos: **R2adj**, **AIC**, y **BIC**. Esta instancia de evaluación estadística resulta útil para comprender que *variables independientes* son significativas.

Modelos Clásicos de ML

Aparte de los modelos de regresión mencionados anteriormente, existen otra clase de modelos dentro de la variedad que existen, que sirven para **clasificación** pero también para **regresión**, estos pueden ser:

- **Support Vector Regressor**: basado en support vector machine.

- **Decision Tree Regressor**: arboles de decisión aplicado a regresión.
- **K-Nearest Neighbours Regressor**: basado en k-vecinos cercanos.
- **Random Forest Regressor**: al ser un método ensamblado de *arboles de decisión* también aplica para regresión.
- **Redes neuronales**: las redes neuronales es su unidad fundamental (neurona) están asociadas con una regresión que puede ser del tipo lineal o no lineal.

Para esta etapa de **regresión multivariante**, regresión con muchas características, solo veremos los primeros dos modelos:

1. **Support Vector Machines** (aplicado a regresión)**.**
2. **Decission Tree Regressor.**

Support Vector Machines

Algoritmo de **aprendizaje supervisado** desarrollado por Vladimir Vapnik y su equipo en los laboratorios AT&T.

Es un método de **aprendizaje supervisado** para la resolución de problemas de **clasificación** y **regresión**.
El algoritmo de SVM aplicado a la regresión numérica se conoce como **regresión de vectores de soporte** o **SVR**.

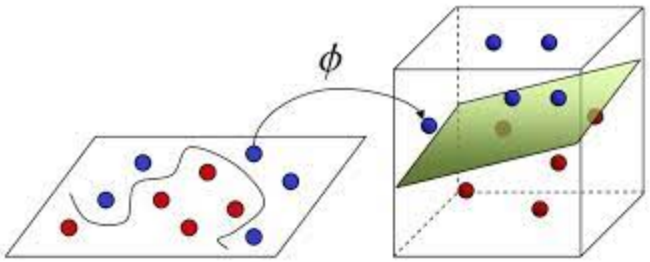
En la regresión de vectores de soporte, la línea recta que se requiere para ajustar los datos se denomina **hiperplano**. La idea básica detrás de SVR es encontrar la mejor línea de ajuste cómo en **OLS**.

Entonces la línea de *mejor ajuste* es un hiperplano que tiene el número máximo de puntos.

Los puntos de datos a cada lado del hiperplano que están más cerca del hiperplano se denominan v**ectores de soporte**. Estos influyen en la posición y orientación del hiperplano y por lo tanto ayudar a construir la SVM para regresión.

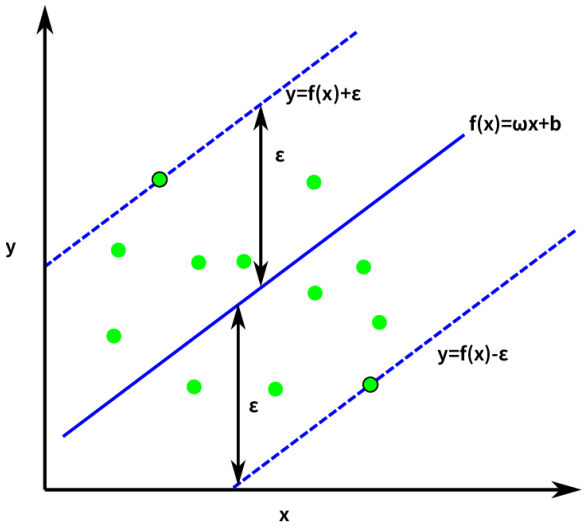
Puede ocurrir que los datos no sean lineales. Por lo que. en el caso de que no puedan ajustarse por una función lineal, se recurre a la metodología denominada **Kernelización** (**kernel**). Esto quiere decir que, los datos pertenecientes al **espacio original** de entradas se transforman en un **nuevo espacio** en el que sí es posible ajustar los datos transformados mediante un regresor lineal.

El tipo de transformación dependerá del tipo de función kernel utilizado, pudiendo ser: Polinomial, Sigmoidea, Función de Base Radial (**RBF**), entre otras.



Hiperparametros

1. Hiperplano: Los hiperplanos son límites de decisión que se utilizan para predecir la salida continua. Los puntos de datos a ambos lados del hiperplano que están más cerca del hiperplano se denominan vectores de soporte. Estos se utilizan para trazar la línea requerida que muestra la salida prevista del algoritmo.



2. Kernel: Es un conjunto de funciones matemáticas que toma datos como entrada y los transforma en la forma requerida. Estos se utilizan generalmente para encontrar un hiperplano en el espacio dimensional superior. Los núcleos más utilizados incluyen **lineal**, **no lineal**, **polinomial**, función de base radial (**RBF**) y **sigmoide**. De forma predeterminada, RBF se utiliza como núcleo. Cada uno de estos núcleos se utiliza según el conjunto de datos.

3. Líneas límites: Estas son las dos líneas que se dibujan alrededor del hiperplano a una distancia de ε (épsilon). Se utiliza para crear un margen entre los puntos de datos.
4. C: es la inversa a la fuerza de regularización es otro hiperparámetro que podemos ajustar.
   A medida que C aumenta, nuestra tolerancia para los puntos fuera de ϵ también aumenta. En cambio si C se aproxima a 0, la tolerancia se aproxima a 0 y la ecuación se reduce a la forma más simplificada (aunque a veces inviable en la práctica).
5. Epsilon: Especifica el tubo épsilon dentro del cual no se asocia ninguna penalización en la función de pérdida de entrenamiento con puntos predichos dentro de una distancia épsilon de la real valor.

> 💡 ¿Qué pasa si solo nos preocupamos por reducir el error hasta cierto punto? ¿Qué pasa si no nos importa cuán grandes son nuestros errores, siempre y cuando estén dentro de un rango aceptable?

A diferencia de otros modelos de regresión que intentan minimizar el error entre el valor real y el predicho, el SVR intenta ajustar la mejor línea dentro de un valor de umbral.

OLS, Lasso y Ridge son todas extensiones de una simple ecuación $y = mx+b$, con un parámetro de penalización adicional que apunta a minimizar la complejidad y/o reducir la cantidad de funciones utilizadas en el modelo final. Independientemente, el **objetivo** es reducir el error del conjunto de prueba.

Tomemos como ejemplo la predicción de precios de viviendas. ¿Qué pasa si estamos de acuerdo con que la predicción esté dentro de una cierta cantidad de dólares, digamos $ 5,000? Luego podemos darle a nuestro modelo cierta flexibilidad para encontrar los valores predichos, siempre que el error esté dentro de ese rango.

SVR nos brinda esta flexibilidad de definir **cuánto error** es **aceptable** en nuestro modelo y encontrará una línea apropiada (o hiperplano en dimensiones más altas) para ajustar los datos.

Esto lo logra estableciendo un valor de umbral (**epsilon**) que la distancia entre el hiperplano de la *linea regresora* y la línea límite.

SVR con un kernel *lineal* proporciona una implementación más rápida que SVR pero solo considera funciones lineales.

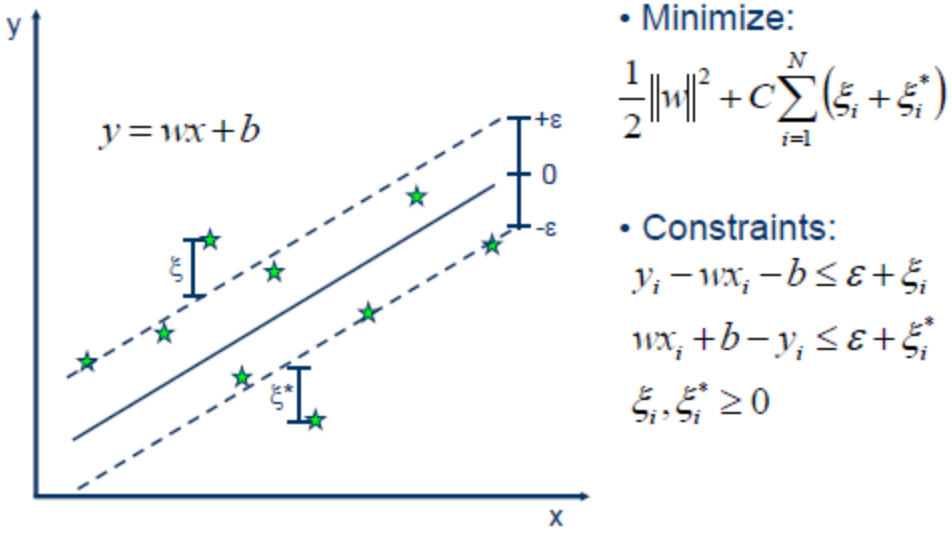
A diferencia de OLS, la función objetivo de SVR es minimizar los coeficientes, más específicamente, la norma w2 del vector de coeficientes, no el error cuadrático.

$$MIN\ \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2$$

Donde **W** es la función de regularización (penalización L2). Al disminuir está es que se produce que se deja un error, ya que no se ajusta tan fuertemente como si estuviéramos penalizando los parámetros de la función lineal. De esta forma se obtiene un **margen máximo**. Por lo que podemos decir que **W** es la magnitud de aproximación a los márgenes máximos.

Es margen máximo se conoce como ***E-tube***. Está formado por los **vectores de soporte**, es decir aquellos puntos que *entran* en la tolerancia que me permite la sensibilidad del error (**E-insensitive**).

El **E-sensitive** maneja dos cosas: el error en la predicción y a la vez que tan estrecho será el **E-tube**. Es decir con que tolerancia de error serán las predicciones.



Ventajas de SVR

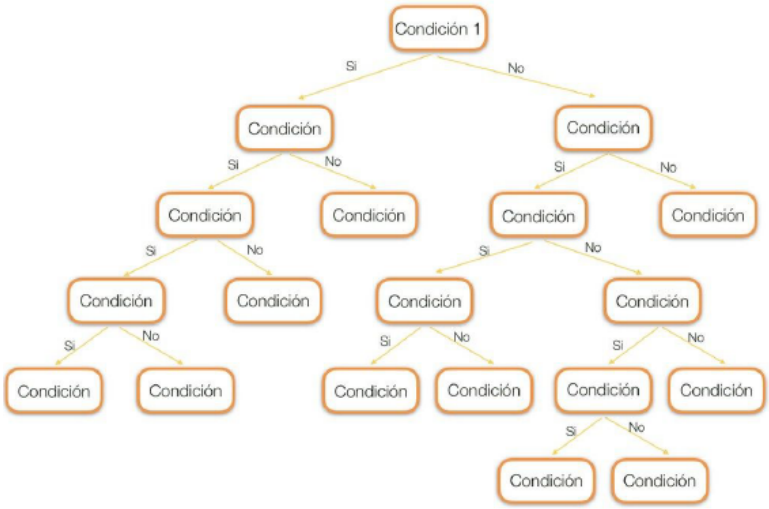
- Su dependencia de relativamente pocos vectores de soporte significa que son modelos muy compactos y ocupan muy poca memoria.
- Una vez entrenado el modelo, la fase de predicción es muy rápida.
- Debido a que solo se ven afectados por puntos cercanos al margen, funcionan bien con datos de alta dimensión, incluso datos con más dimensiones que muestras, lo cual es un régimen desafiante para otros algoritmos.
- Su integración con los métodos de Kernel los hace muy versátiles, capaces de adaptarse a muchos tipos de datos.

Desventajas de SVR

- Para un gran número de muestras de entrenamiento, este costo computacional puede ser prohibitivo.
- Los resultados dependen en gran medida de una elección adecuada para el parámetro de suavizado C. Esto debe elegirse cuidadosamente a través de la validación cruzada, lo que puede ser costoso a medida que los conjuntos de datos aumentan de tamaño.
- Los resultados no tienen una interpretación probabilística directa. Esto se puede estimar a través de una validación cruzada interna, pero esta estimación adicional es costosa.

Decision Trees

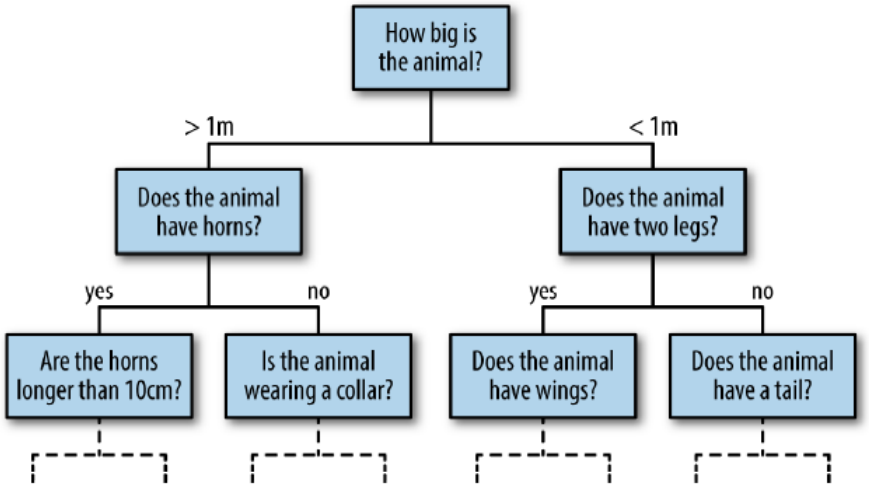
Decision Tree es un algoritmo clásico para resolver problemas. Intenta simular el proceso de pensamiento lógico humano binarizando cada paso de la decisión. Por lo que, en cada paso, el algoritmo elige entre Verdadero o Falso para avanzar.



Es un algoritmo **no-paramétrico**, es decir si bien tiene parámetros que controlan su entrenamiento, no tiene que **aprender parámetros fijos** de una función lineal como en los modelos anteriores. Por ejemplo un parámetro es el tamaño del árbol, este al principio no está definido y puede variar dependiendo de los datos de entrada.

Es un algoritmo simple, pero muy poderoso, por lo que se aplica ampliamente en modelos de aprendizaje automático. Sin embargo, uno de los problemas de los árboles de decisión es su dificultad para generalizar un problema. El algoritmo aprende tan bien cómo decidir sobre un conjunto de datos dado que cuando queremos usarlo para nuevos datos, no nos da la mejor respuesta. (spoiler alert: overfitting!).

Los **árboles de decisión** son un forma extremadamente intuitiva de clasificar o etiquetar un objeto, solamente se realizan una serie de preguntas muy sencillas diseñadas para lograr el objetivo.



En un árbol bien construido, cada pregunta reducirá la cantidad de opciones a aproximadamente la mitad, reduciendo muy rápidamente las opciones incluso entre una gran cantidad de clases. El truco, por supuesto, está en decidir qué preguntas hacer en cada paso.

En una implementación de árbol de decisión, las preguntas generalmente toman la forma de divisiones alineadas con ejes en los datos; es decir, cada nodo del árbol divide los datos en dos grupos utilizando un valor de corte dentro de una de las características.

Terminologías:

- Nodo Raíz: representa toda la población de los datos que tenemos o existen. Este nodo se puede dividir en dos o más conjuntos.
- División: es el proceso de dividir un nodo en dos o más subnodos.
- Nodo de decisión: cuando un nodo se divide en dos o más nodos, al inicial lo llamamos Nodo de decisión.
- Nodo Hoja: aquellos nodos que no se dividirán llamados nodos terminales o nodos hoja.
- Rama: una subsección de todo el árbol se conoce como sub-árbol o rama.
- Principal y secundario: un nodo que se divide en sub-nodos se denomina nodo principal y los sub-nodos se denominan nodos secundarios.

- Profundidad del árbol: es la longitud del camino más largo desde el nodo raíz hasta el nodo hoja.

Los árboles de decisión tienen la ventaja de que son fáciles de entender, requieren una mínima limpieza de datos, la no linealidad no afecta el rendimiento del modelo y la cantidad de hiperparámetros para ajustar es muy baja. Sin embargo, puede tener un problema de sobreajuste, que se puede resolver utilizando el algoritmo **Random Forest** .

¿Cómo se elige la forma de hacer las particiones?

- Recursivo: Divide el trabajo en partes, y resuelve cada parte diviendolas a su vez en partes más pequeñas.
- Voraz (greedy): en cada paso de la construcción del árbol se busca la mejor división en ese punto en particular en lugar de mirar hacia adelante y elegir una división que llevaría a una mejor árbol en un paso futuro.
- Óptimo local: no alcanza la mejor solución de todas las posibles, sino una solución localmente óptima.

Optimización del rendimiento del árbol de decisión

1. **Sobreajuste**: quizás la división seleccionada no sea la mejor y terminaremos con un gran árbol de decisión. Esto puede tener un rendimiento excelente con los datos del tren (mucho ajuste), pero no tan bien con los datos no vistos. En este caso podemos:
    a. Restringir la profundidad de los nodos de decisión.
2. **Subajuste**: cuando el modelo no se ajusta tan bien a los datos del tren.
    a. Establezca el número mínimo de muestras de nodos. No más divisiones más allá de este mínimo.
    b. Establezca el número mínimo de muestras para un nodo terminal. Muy similar al anterior pero con las muestras dentro (los valores altos controlan el sobreajuste, los valores demasiado altos pueden provocar un ajuste insuficiente).
    c. Establezca la profundidad del árbol (una mayor profundidad puede provocar un ajuste excesivo, una profundidad más baja puede provocar un ajuste

insuficiente).

> El sobreajuste resulta ser una propiedad general de los árboles de decisión; es muy fácil profundizar demasiado en el árbol y, por lo tanto, ajustar los detalles de los datos particulares en lugar de las propiedades generales de las distribuciones de las que se extraen.

Referencias

1. VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data*. " O'Reilly Media, Inc."
2. Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.
3. Certificación Universitaria en Data Science. Mundos E - Universidad Nacional de Cordoba.
4. Certificación en Machine Learning - Machine Learning for Beginners. Analytics Vidhya.

Documentación extra-curricular

https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2

https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0

https://towardsdatascience.com/machine-learning-basics-decision-tree-regression-1d73ea003fda

Made with ❤️ Ignacio Bosch