# Modelos predictivos con machine learning

## Índice

# Semana 1

## Introducción a la semana 1

Actualmente el aprendizaje automático, o comúnmente conocido por su nombre en inglés "Machine Learning" es un área que cuenta con mucho interés dentro de la comunidad de científicos de datos debido a la facilidad que existe para implementar sus técnicas para resolver diferentes problemas del mundo real, lo cual ha aumentado mucho la demanda de los científicos de datos en diferentes ámbitos laborales y profesionales.

Juntos analizaremos diferentes tipos de modelado de datos con la finalidad de que al terminar esta semana seas capaz de identificar cuál es el modelo que mejor se ajusta a las necesidades de tu problema y de igual manera estudiaremos los diferentes tipos de técnicas dentro del área de aprendizaje reforzado.

En esta sección se abordarán múltiples conceptos y temas que se complementarán paulatinamente a través de prácticas con Python durante el resto de ese curso y no solamente durante esta semana. Recuerda estudiar las cosas a tu ritmo a tu tiempo y siempre que te través volver a leer una y otra vez el material.

## Explorando Jupyter

"Jupyter Notebook". Entonces la vamos a abrir la aplicación. Va a abrir esta ventanita. Vamos a espera unos momentos. A veces es necesario dar un Enter. Entonces, nos va a mandar a esta ventana. Esta es la ventana principal de Jupyter.

Aquí es donde vamos a poder seleccionar nuestros cuadernos de trabajo. Tenemos nuestros archivos, nuestros cuadernos que están abiertos actualmente, y podemos cerrar sesión o cerrar el programa. Vamos a generar un programa nuevo.

Vamos a darle New/Python 3. Ok. Esta es nuestra interfaz. Esta es la que vamos a estar utilizando constantemente.

Vamos a ver cada una de las opciones. Primero tenemos este, el logo de aquí, Jupyter, que nos lleva de regreso a nuestra página de acá. Por el momento no lo vamos a seleccionar.

Aquí tenemos el título de nuestro cuaderno. Aquí es donde podemos renombrar. Entonces, vamos a ponerle "Prueba" y vamos a darle un Enter

Entonces ahora nuestro archivo ya se llama "Prueba" y si lo vemos por acá se llama prueba.ipynb, "i Python Notebook".

¿Ok? Tenemos por aquí la pestaña File que nos va a dar opciones como crear un nuevo cuaderno, abrir un cuaderno que ya tengamos hecho, hacer una copia de nuestro cuaderno para hacer un duplicado. Podemos guardar el cuaderno, renombrarlo, que esta función es la misma de dar clic acá arriba.

Podemos guardar un estado actual para poder regresar a ese estado que se hace de esta forma. Lo acabamos de crear. Podemos imprimirlo, el cuaderno. Podemos descargarlo en algún otro formato. Podemos cerrarlo.

Tenemos la pestaña Edit en la que igual tenemos opciones como: cortar celdas, copiar celdas, pero en este momento se podrán preguntar ¿qué es una celda?. Bueno, una celda es el elemento principal de trabajo. Es el que vamos a tener en nuestra área, aquí de Jupyter, en la cual podemos escribir código de Python y ejecutarlo.

También podemos tener Markdowns, los cuales nos permiten escribir texto. Si ponemos un símbolo de gato (#) podemos tener títulos, o incluso podemos tener texto común, el cual nos va a permitir hacer anotaciones en nuestro código.

Una vez establecido qué es una celda, podemos empezar a hacer operaciones sobre las celdas. Por ejemplo, podemos copiarla. Copiamos esta celda de arriba y la podemos pegar. Entonces, se hace el copiado de todo esto y se pega aquí. Entonces, tenemos Pegar, tenemos Eliminar, tenemos Cancelar la eliminación de la celda, podemos dividir una celda, podemos combinar celdas, moverlas, cambiar el orden de ellas, encontrar algunas celdas y poder reemplazar elementos de la celda. Incluso eliminar los adjuntos de la celda.

Esto lo veremos más adelante cuando veamos Graficación.

También tenemos la pestaña View. Podemos activarla y veremos que se nos empiezan a ocultar herramientas. Por el momento vamos a dejarlo tal como está.

También tenemos la Insert que nos va a ayudar a insertar celdas arriba y abajo de donde estamos seleccionando.

Tenemos la pestaña Cell, que nos va a permitir ejecutar nuestras celdas. Por ejemplo, si pongo el cursor sobre "print("Hola")" y le doy Run Cel, la va a ejecutar rápidamente. El número de ejecución de celda se reacomoda a la parte izquierda.

También podemos correr todas las celdas al mismo tiempo. Podemos correr las celdas a partir del cursor hacia arriba con Run All Above y de esa celda hacia abajo con Run All Below. Podemos interrumpir nuestro Python, podemos reiniciarlo y podemos apagarlo en caso de que algo salga mal con nuestro código, que quedemos atorados en algún ciclo infinito. Alguna situación similar.

También tenemos la pestaña de Help que nos va a ayudar en cualquier duda que tengamos a poder darnos una ayuda rápida. Por ejemplo, aquí está el tour de Python que lo pueden ir viendo un paso a la vez.

Después de ver la barra principal de ayuda, tenemos también los botones que están en la sección un poquito más abajo. Nos vamos a ir de izquierda a derecha. Tenemos el Guardar, tenemos el Crear una nueva celda, tenemos Cortar, Copiar y Pegar celdas, tenemos Mover la celda hacia arriba o mover la celda hacia abajo, podemos ejecutar las celdas individualmente, detener nuestro kernel, reiniciar nuestro kernel y reiniciar nuestro kernel y ejecutar todo nuestro cuaderno.

También tenemos esta parte de aquí que nos va a permitir cambiar nuestras celdas entre código y markdown.

Y además tenemos este pequeño icono aquí, que es una paleta de comandos, el cual nos tiene todos los posibles comandos de Python, y para usarlo simplemente escribimos lo que queremos. Por ejemplo, si escribimos la palabra "move", de mover, nos da recomendaciones: mover celdas hacia abajo, mover celdas hacia arriba, mover el cursor hacia abajo o mover el cursor hacia arriba. Entonces cualquier cosa que quieran hacer y se les olvide pueden utilizar este botón para buscar la opción indicada.

Además, acá a la derecha, tenemos un círculo que por el momento está blanco. Eso indica que el kernel de Python, o sea, el motor que nos va a estar ejecutando nuestras celdas, ahorita está descansando. Cuando realizamos una acción se pondrá rápidamente en negro. Cuando termine se va a pasar a blanco. Eso indica que mandamos una carga de trabajo al motor de Python y la ejecutó. Entonces, si tenemos una carga muy pesada que no termina, ese círculo va a estar en negro.

Es un indicador que les va a ayudar a saber el estado del motor. Entonces, con eso terminamos de ver nuestra interfaz de Jupyter.

Esta es en la que vamos a estar trabajando en la mayoría del curso, y con eso vamos a ejecutar los demás cuadernos.

## Introducción al modelado de datos

El modelado de datos nos ayuda a describir comportamientos que suceden en la realidad utilizando ecuaciones matemáticas. Lo anterior se logra a partir de la observación de las relaciones causa y efecto entre dos o más variables.

Desde la invención del plano cartesiano, el ser humano ha utilizado esta herramienta para visualizar fenómenos en la naturaleza mediante el modelado de datos, lo que ha llevado a un avance tecnológico y científico, además de otorgarnos la capacidad de predecir y controlar eventos.

El modelado de datos es el proceso que permite construir o identificar una representación abstracta de la información observada con la finalidad de analizar características importantes dentro de los datos e intentar describir el comportamiento de dichos datos.

En esta unidad te introduciremos a los conceptos fundamentales relacionados con el modelado de datos, explicaremos algunos de los diferentes tipos de modelado de datos e identificaremos los elementos que nos permitan escoger adecuadamente el modelo que se adecue mejor a diferentes tipos de problemas.

Además, utilizaremos nuestros modelos para intentar predecir cómo se comportarán nuestros problemas ante situaciones novedosas o diferentes a las que se estaba acostumbrado.

## Identificando los tipos de modelos

Debes ser tener conocimiento de los tipos de datos con los que trabajaremos, esto nos ayudará a elegir el algoritmo de aprendizaje que corresponda con nuestro objetivo.

Empezaremos a manejar lo que es el modelado en ciencia de datos, pero, como preámbulo, vamos a empezar a utilizar algunos consejos que les recomendamos que empiecen a utilizar antes de entrar al modelado.

Para ello vamos a utilizar un cuaderno de Jupyter. Para abrirlo tenemos que dar inicio y escribimos Jupyter, damos Enter y esperamos unos segundos a que cargue.

Va a cargar en esta carpeta y van a tratar de ir a donde hayan descargado el material del curso. En nuestro caso lo tenemos en el escritorio, entonces vamos a escritorio, cuadernos y aquí está la carpeta que buscamos.

Para esta sección en específico vamos a utilizar el cuaderno "Identificar el tipo de modelado", o de modelo, para nuestro problema. Al abrirlo nos contiene este texto.

Entonces, antes de poder empezar a modelar nuestro problema hay que identificar específicamente qué tipo de problema tenemos y para ello primero, lo primero que hay que hacer es identificar nuestros datos, qué es lo que tenemos, qué es lo que queremos y cómo empezar a comprender qué es lo que tenemos más allá de "tenemos una tabla", sino, qué es lo que representa la información en esa tabla.

Entonces, para empezar a analizar esa información le sugerimos utilizar en primer instancia el método "describe( )" que esté incluido dentro de Pandas. Este nos va a decir... pues los promedios, los mínimos, los máximos, los cuartiles, el conteo de registros, entonces nos empieza a dar una idea de la estructura de nuestra formación y de la forma de nuestros datos.

Derivado de esto de "describe( )" nos va a decir la media y la mediana. Esto nos ayuda a identificar la tendencia central de nuestros datos, es decir, dónde está la mayoría de nuestra información. También tener mucho cuidado con los percentiles. Los percentiles nos va a decir mas o menos el rango de nuestros datos para darnos una idea a nosotros como analista de datos cómo se están estructurando.

Además les recomendamos hacer una matriz de correlación, esto nos va a empezar a dar una idea de qué pares de variables trabajan en conjunto o tienen una relación entre sí. Ahora, correlación no es causalidad, tengan mucho cuidado. No porque dos variables tengan una correlación alta significa que son dependientes una de la otra. Puede darse la casualidad que crecieron similarmente pero una no causa la otra y viceversa.

También les recomendamos hacer visualizaciones, las visualizaciones les van ayudar a entender más allá de números, les va a ayudar gráficamente a analizar la información y entender la distribución de los datos.

Por último les recomendamos, una vez que hayan analizado más o menos cómo se distribuye su información, hacer limpieza de datos anómalos, es decir, limpiar datos faltantes y limpiar datos outliers que no representan a su población en general.

Una vez que analizamos nuestra información, sabemos lo que tenemos y qué es lo que representa podemos ir a la librería scikit-learn es la que vamos a estar utilizando al menos en este curso y ya viene dentro de nuestra instalación de anaconda.

Esto nos va a permitir modelar de una forma sencilla todo lo que queramos relacionado con Machine Learning que ya tiene todo incluído. Entonces nosotros vamos a tomar los modelos de ahí

y los vamos a ejecutar con la información que nosotros le pongamos, y a partir de ahí vamos a poder obtener modelos concisos y de simple distribución.

Scikit-learn nos ofrece esta hoja o este diagrama el cual nos permite hacernos una idea, ya una vez con nuestro conjunto de datos, de qué tecnología deberíamos utilizar dentro de su librería.

Vamos a analizar un poco. Inicia por donde dice Start, y nos empieza a dar unas preguntas de "sí" o "no". La primera recomendación que nos hacen ellos es que si tienes menos de 50 registros pues vayas a conseguir más porque no vas a poder realizar modelos confiables con tan poquita información.

A partir de ahí nos pregunta "¿lo que buscas predecir es una categoría?". Si tu respuesta es que sí ya hay dos tipos de modelos diferentes que quieres hacer. Si la información que tienes ya está etiquetada estás buscando lo que es una clasificación y está representada por todo este círculo rosa (Figura 1).

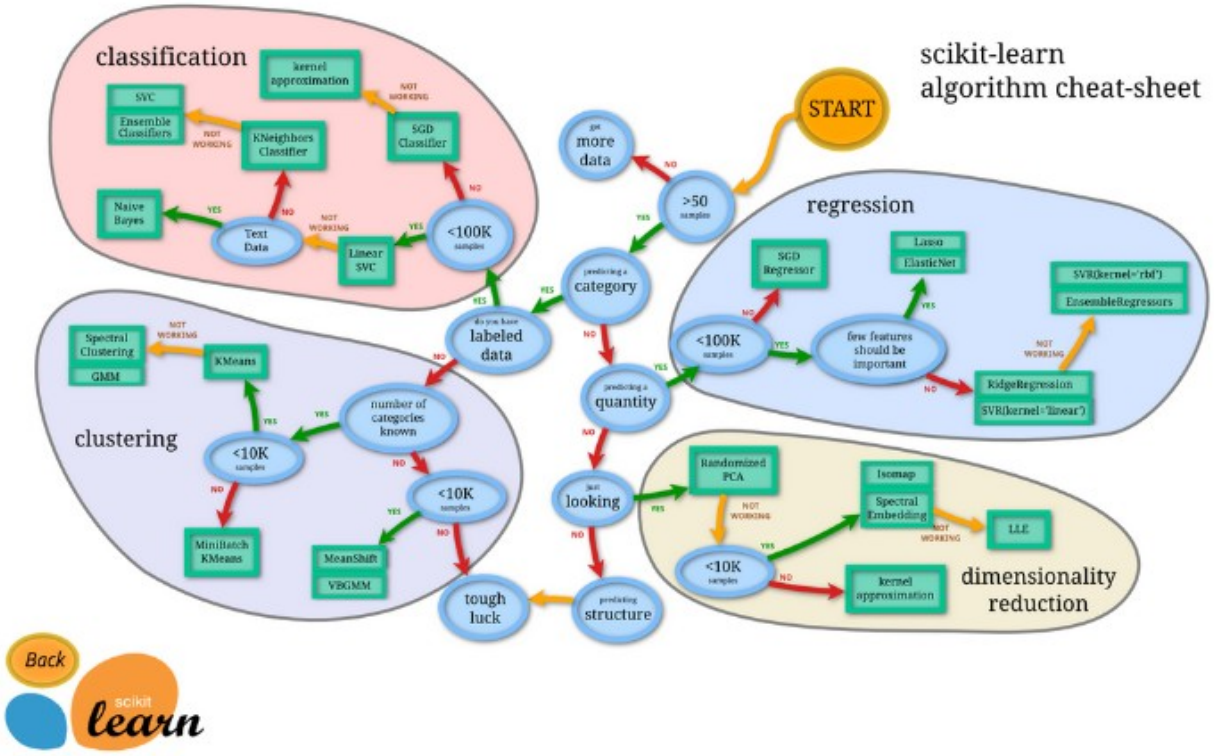


*Figura 1: Scikit-learn cheatsheet*

Si esta información no está etiquetada lo que estás tratando de hacer es sepreparar la información en grupos, es decir, estás haciendo una agrupación que en inglés se llama "clustering", esto es todo en una otra área del Machine Learning-

Si no estabas prediciendo una categoría tal vez estabas buscando predecir una cantidad, tal vez la temperatura de mañana o la calificación para un examen, esos son valores numéricos y son cantidades. Si este es el caso pues ya estás enfrentándote a problemas de "regresión". "Regresión" lo que busca es predecir es un número, y esto es representado por este círculo azul.

En caso de que no sea eso lo que en realidad estás buscando es tratar de simplificar la información de alguna manera. Esto nos lo ofrece scikit-learn con este círculo café, que es, reducir la

dimensionalidad del problema, es decir, tomar cinco días variables y reducirla tal vez a dos o una. Esto es algo que nos ofrece.

De esta figura podemos extraer algo de información, algo ya más conciso. Nuestro problema se va a identificar sobre todo por lo que estamos buscando, o por nuestro tipo de salida. Es decir, si nosotros buscamos predecir una clase lo que tenemos es un problema de clasificación, si queremos producir un número tenemos un problema de regresión y si queremos separar los datos tenemos un problema de agrupamiento.

En particular en este curso no vamos a ver o enfatizar tanto en reducción de dimensionalidad. ¿Por qué? Porque estos son los tres grandes problemas de Machine Learning: clasificación, regresión y agrupamiento. Vamos a dar una muy leve introducción a lo demás, pero esto lo que vamos a estar trabajando.

Además, podemos hacer una segunda categoría de nuestros problemas y esto es por el tipo de datos de entrada que estamos recibiendo. Si tenemos datos etiquetados tenemos un problema de aprendizaje supervisado. ¿Qué representa el aprendizaje supervisado? Bueno, que con nosotros, como podemos supervisar cómo estaba funcionando el algoritmo, a través de estas etiquetas diciéndole "te equivocaste o no te equivocaste".

Si tenemos datos no etiquetados tenemos un aprendizaje no supervisado. ¿Por qué? Porque no tenemos en realidad una respuesta correcta, solamente estamos esperando a ver cómo agrupa el algoritmos los datos y poder nosotros realizar conclusiones a partir de ello.

Si queremos realizar un aprendizaje donde la computadora esta interactuando con un ambiente. Lo que tenemos es aprendizaje reforzado, esto es lo que se utiliza generalmente, por ejemplo, cuando hay inteligencia artificial en videojuegos.

No hay alguien que haya entrenado manualmente a la computadora. Por lo general a manera de prueba y error estuvo aprendiendo las reglas del juego, y así puedo aprender a ser bueno en él. En posteriores videos vamos a enfatizar lo que es Machine Learning, y a expandir un poco esa definición, y vamos a empezar a crear modelos para poder predecir a futuro.

De la figura anterior también podemos extraer una sencilla clasificación de los tipos de problemas que vamos a abordar dependiendo lo que buscamos obtener como nuestra salida. Si queremos:

- Predecir una clase, tenemos un problema de clasificación.
- Predecir un valor numérico, tenemos un problema de regresión.
- Separar los datos en diferentes grupos, tenemos un problema de agrupamiento.

Además, también podemos definir el tipo de aprendizaje a realizar definido por los datos de entrada.

- Si tenemos datos etiquetados, tenemos un problema de aprendizaje supervisado.
- Si tenemos datos no etiquetados y queremos encontrar una estructura, tenemos un problema de aprendizaje no supervisado.
- Si queremos aprender a partir de la interacción con un ambiente, tenemos aprendizaje reforzado.

# Fundamentos de Machine Learning

Los modelos de Machine Learning son diversos, pero efectivos una vez que se identifica cuál es el que mejor se ajusta a nuestra base de datos.

En esta sección aprenderás todos los conceptos fundamentales y todas las bases que necesites para entender qué es el aprendizaje automático. Esta sección está centrada en que conozcas el concepto general y algunos de los algoritmos más utilizados. Para eso hemos preparado una libreta donde viene toda la teoría fundamental de esta área, pero no creas que solo exploraremos esto teóricamente.

Al menos en esta sección solamente veremos la teoría y te introduciremos a cada uno de estos algoritmos, pero esta semana y en posteriores veremos uno a uno cómo se va ejecutando cada uno de estos.

Para esto vamos a la carpeta donde tengas todos tus cuadernos y abre el cuaderno de machine learning. Aquí en pantalla estamos viéndolo.

Vamos a leer juntos la primera parte. Dice: el área de aprendizaje automático se destaca por su vital importancia dentro de la carrera de un científico de datos. El aprendizaje automático, mejor conocido por su término en inglés "machine learning", busca desarrollar software capaz de reconocer patrones dentro de un conjunto de datos para posteriormente realizar un aprendizaje y por último poder construir modelos que representan a dicho conjunto de datos.

El aprendizaje automático puede resultar de ayuda a la hora de usar datos históricos para tomar mejores decisiones de negocios o decisiones fundamentales basándose en acontecimientos pasados de una persona o de una empresa. En esta libreta te explicaremos brevemente algunos de los algoritmos más populares de esta área. Así que vamos a empezar.

## Regresión lineal

El primer algoritmo que tenemos aquí es el de la regresión lineal. La regresión lineal es un método matemático que modela la relación entre una variable independiente, usualmente asociada a la letra "x", y una variable dependiente usualmente asociada a la letra "y", por medio de una expresión lineal. Para los que no están familiarizados con las expresiones lineales son ecuaciones de matemáticas que al momento de graficarlas en un plano cartesiano nos representan una línea recta, y esta línea tiene la característica que va a intentar pasar por la mayor cantidad de los datos posibles tratando de asemejarse lo más posible a la representación de los datos.

En pantalla podemos observar que hay muchos puntos azules. Estos puntos azules son nuestros datos. La regresión lineal construye la recta naranja que esta graficada que atraviesa la mayoría de estos puntos.

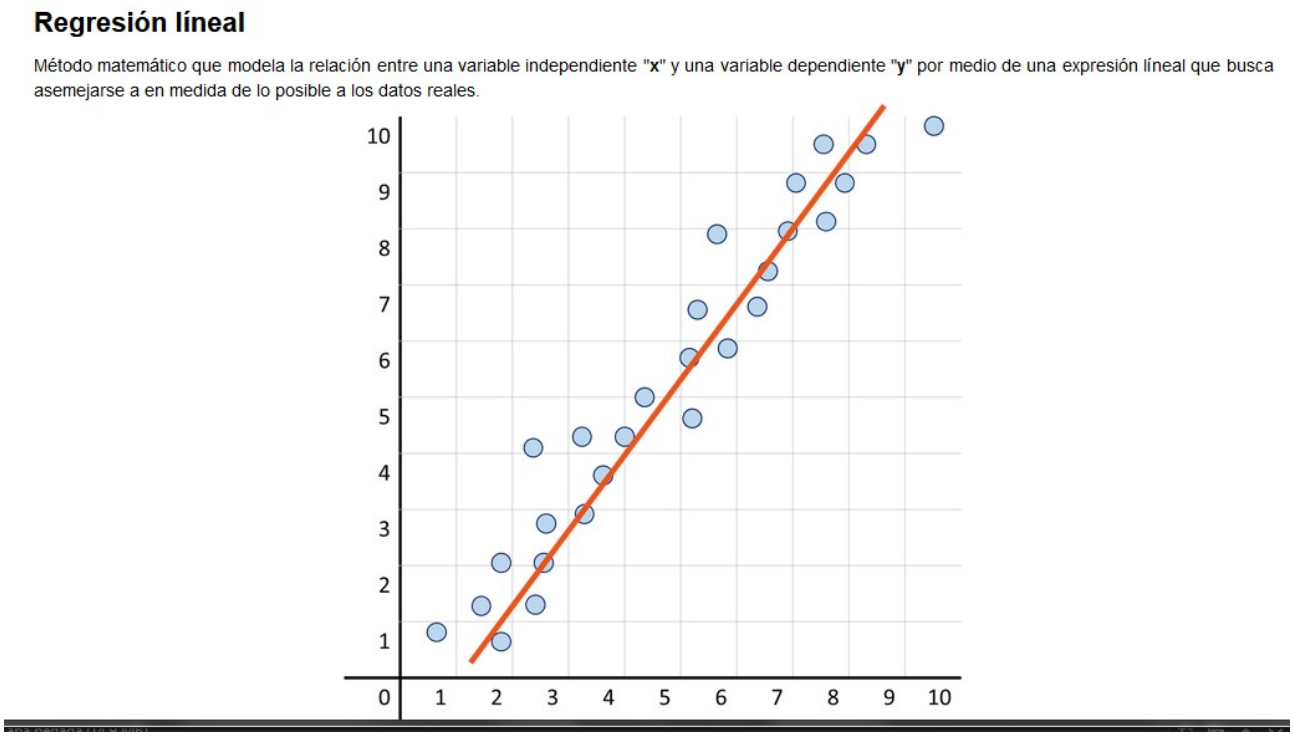

*Figura 2: Regresión lineal*

## Regresión polinomial

El siguiente algoritmo del que vamos a hablar es la regresión polinomial. La regresión polinomial es muy parecida a la regresión lineal, simplemente que extiende a la función "f=mx + b", que es la fórmula de una función lineal, al terreno de un polinomio. En este terreno ya tenemos un polinomio que es un poco más flexible, se puede curvar cuando es necesario y a medida que uno trabaje con polinomios sabe que si incrementas, por ejemplo el polinomio cuadrado, el cúbico, el cuarto, el quinto, cada vez que vamos incrementando el grado podemos ir flexionando y modelando un poco más a detalle. En pantalla podemos observar que tenemos un conjunto de puntos... un conjunto de puntos mucho más complicado de graficar que el conjunto anterior, y tenemos una línea naranja que representa una curva, esa curva es el resultado de haber aplicado una regresión polinomiales a nuestros datos y graficado el polinomio que mejor se asemeje a las características de nuestros datos.

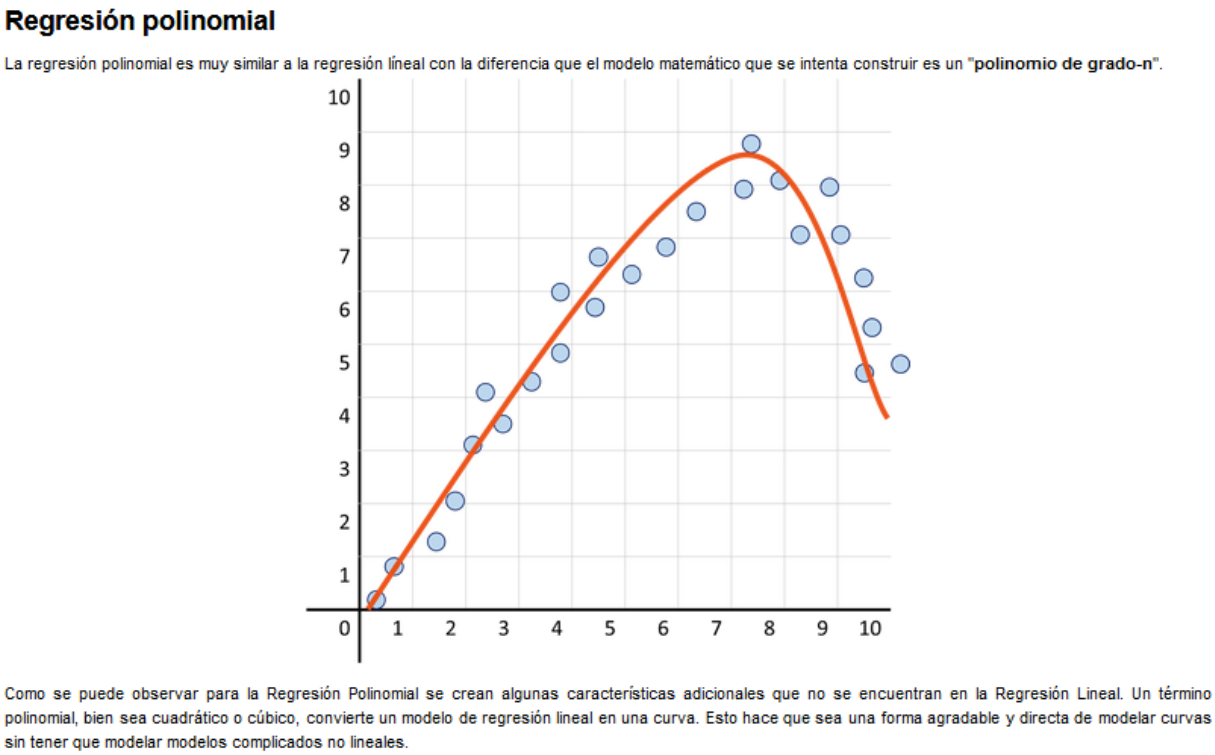


*Figura 3: Regresión polinomial*

## Árboles de clasificación y regresión

¿Qué son los árboles de clasificación? Pues bueno, son unos métodos basados en generar un árbol que intenta explicar o predecir una variable a partir de unas cuantas preguntas o reglas sencillas. En pantalla tenemos cuatro conjuntos distintivos de datos: el conjunto morado, el conjunto amarillo, el conjunto naranja y el conjunto verde, y por ahí tenemos unas cuantas preguntas.

La primera pregunta que encabeza el árbol es: ¿x es menor o igual que 5? Si x es menor o igual que 5 se genera una sección, la sección que está del lado izquierdo, si x es mayor que 5 se genera la sección que está del lado derecho, y cada una de estas dos secciones está partida... ahora esa partida a partir de una "y" ya no de una "x".

Nuestra segunda pregunta que abre más ramas de nuestro árbol, al menos del lado izquierdo, es: ¿"y" es menor o igual que 6? En caso de que sea menor o igual que 6 clasifica el dato como un dato de la región amarilla, sino de la sección morada. De la misma forma podemos seguir explorándolo. A medida que seamos capaces de identificar estas reglas sencillas podemos generar clasificaciones más adecuadas que ahora sí representen de manera más precisa nuestro conjunto de datos.

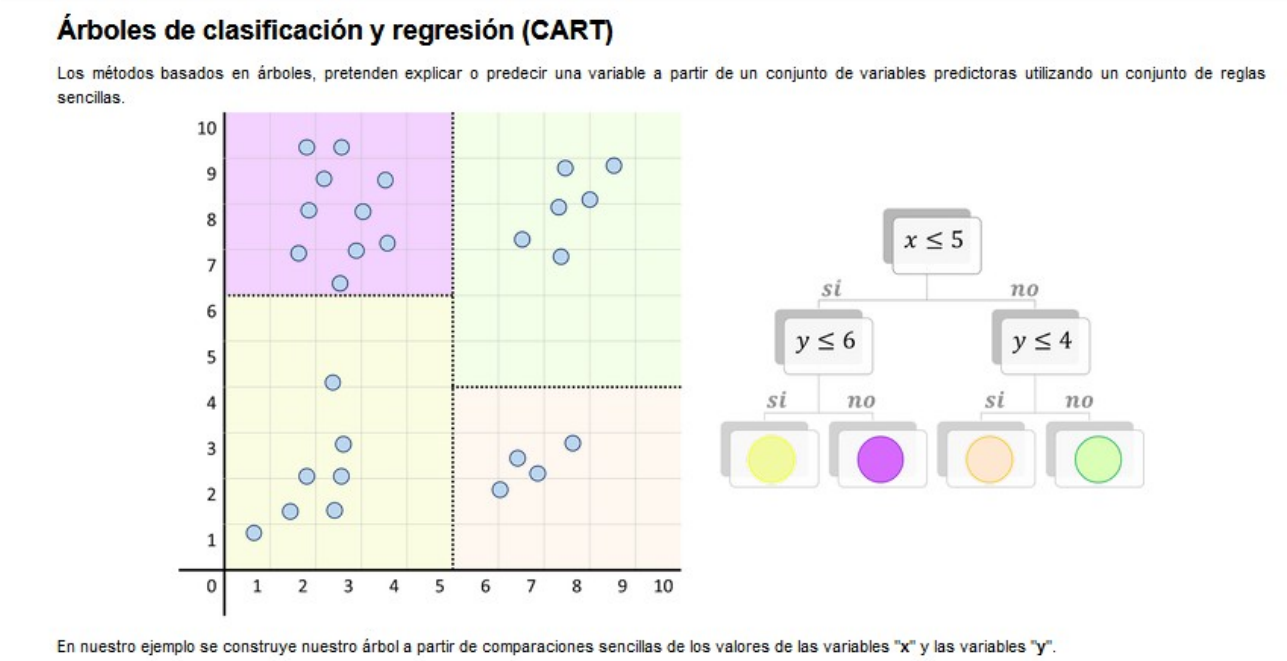


*Figura 4: Árboles de clasificación*

## Redes neuronales

¿Qué son las redes neuronales? Son modelos que están inspirados en la biología, en particular en cómo funciona el cerebro de las personas. Esto significa que están formados por elementos que se comportan de una manera análoga a las neuronas del cerebro, consisten en un conjunto de neuronas artificiales conectadas entre ellas las cuales son capaces de transmitir señales entre ellas y... e ir modificando la información.

Cada vez que la información entre en una neurona se somete a diversas entradas y a diversas operaciones matemáticas que van modificando las entradas hasta obtener valores de salida.

En pantalla tenemos el esquema de una red neuronal. Yo sé que parece que es muchas conexiones, muchos elementos, y verla por primera vez es muy complicado, pero no te preocupes, es un algoritmo muy utilizado, así que existen mucha información de este tema. Pero entonces, hay un conjunto de entradas que pasan a una capa oculta donde se le someten muchas operaciones matemáticas que posteriormente pasan a una capa de salida donde se vuelven a someter unas cuantas funciones matemáticas y producen unas salidas.

Pero qué características benéficas tiene. Este tipo de modelos o algoritmos tienen la característica de que aprenden a partir de la experiencia, generalizan a partir de ejemplos anteriores para buscar reglas que se puedan aplicar en modelos posteriores. A parte abstraen la esencia de los datos de entrada sin necesidad de que uno le describa cuáles son estos datos.

¿Cómo funciona más o menos esto? Yo hago que esta red la construye a partir de un conjunto de datos, la enseño, y después la voy a poder ocupar para predecir nuevos datos.

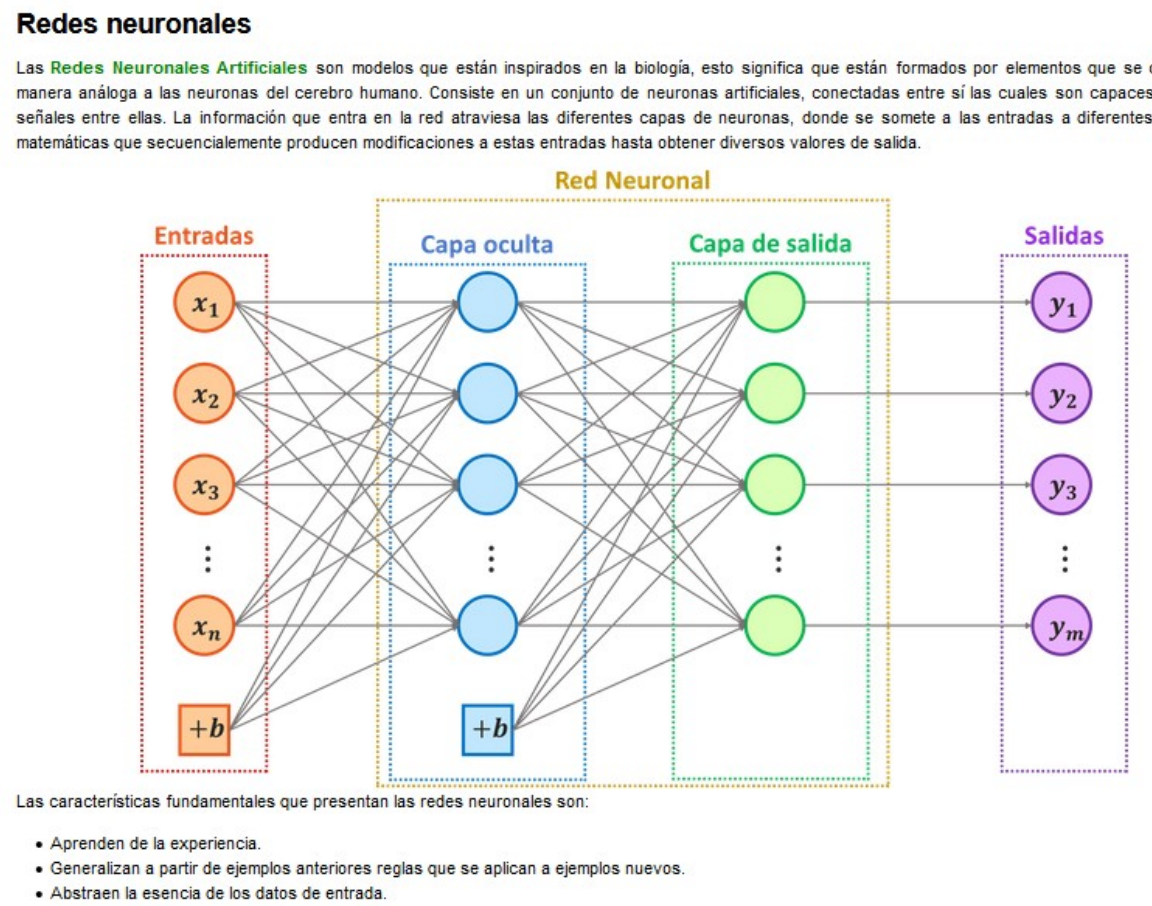


*Figura 5: Redes neuronales*

## Máquinas de soporte vectorial

El último algoritmo que vamos a mencionarte son las máquinas de soporte vectorial. Estas máquinas, o ese algoritmo, buscan encontrar un híper plano que separe de forma óptima a los puntos que están compuestos en una categoría y a los puntos que están compuestos de otra de tal manera que se asemeje a algo parecido a lo que está en pantalla:

tengo una clase que es verde, que todos están muy cerca de los valores de 0,0; tengo una clase morada que está muy lejana del origen. Si yo soy capaz de determinar este hiper plano, es decir, este espacio que separa la clase 1 de la clase 2, yo puedo definir de manera sencilla cuáles son las reglas que generan a la clase 1, la verde, y separan a la clase 2, la morada.

Las máquinas de soporte vectorial crean buscar esto, hacer hiper planos que delimiten con notoriedad cada una de las características de ambos conjuntos.

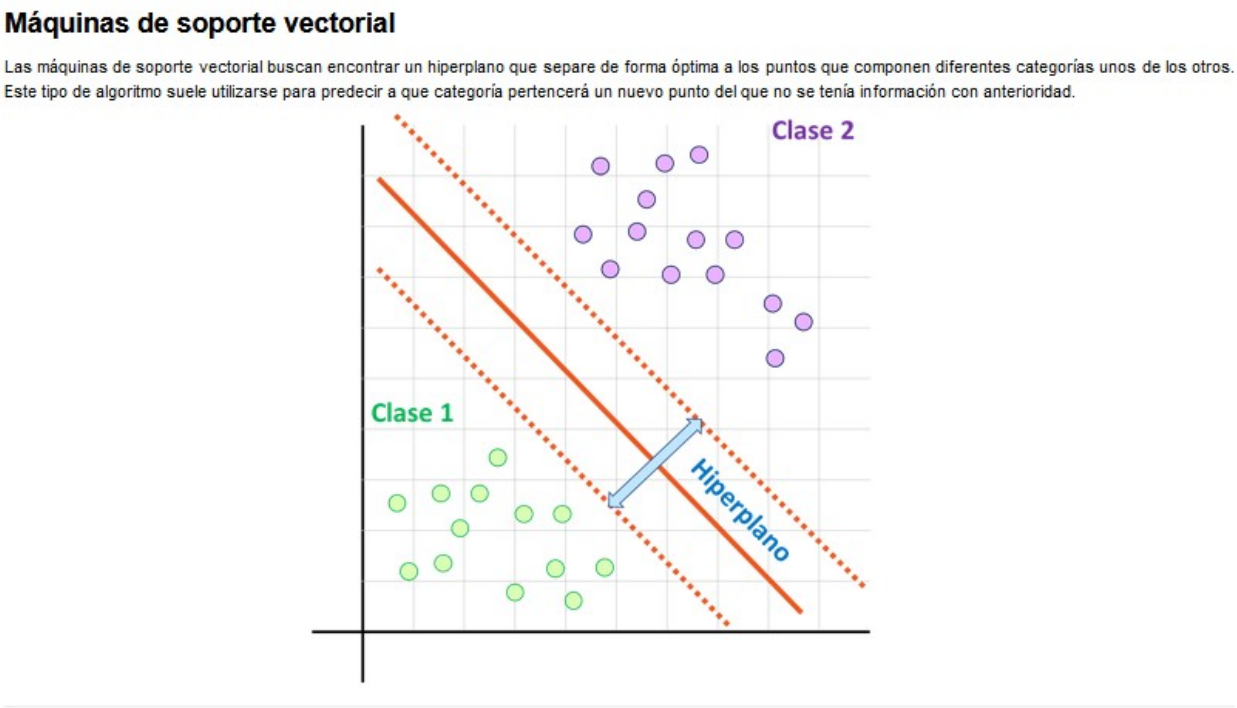


*Figura 6: SVM*

# Tipos de aprendizaje

Existen diferentes tipos de aprendizaje para realizar clasificaciones, agrupamientos y regresiones. Recuerda que dependiendo de nuestra base de datos es el problema de aprendizaje máquina que utilizaremos para realizar predicciones.

En esta sección exploraremos la teoría detrás de los diferentes tipos de aprendizaje que componen el área de aprendizaje automático. Para eso preparamos una libreta de Python que tiene bastante de los conceptos fundamentales que vamos a empezar a introducirte para que sepas lo básico de esta área.

Bueno, nuestra libreta se llama "tipos de aprendizaje". Vamos a la carpeta en la que tú tienes tus cuadernos, en mi caso en el escritorio, y abramos esta libreta juntos. Nuestra libreta, antes de empezar a explorarla, empieza con una definición de qué es inteligencia artificial y el aprendizaje automático para después definir tres de las áreas de aprendizaje que componen esta área.

## Aprendizaje supervisado

Los algoritmos de aprendizaje supervisado son aquellos que trabajan primeramente aprendiendo con un conjunto de datos que previamente nosotros como investigadores les delimitamos como datos de entrenamiento.

Estos datos de entrenamiento tienen la característica de que están etiquetados, es decir, antes de que el algoritmo o en la técnica aprenda ya sabes la solución, ya sabes si es un problema de clasificación, sabes a cada elemento qué clasificación le corresponde.

¿Para qué sirve esto? Para que cuando el algoritmo esté trabajando con ellos él mismo pueda ver si la clasificación que hizo fue correcta y empiezan a modificarse a partir de las etiquetas sin que nosotros le digamos o le modifiquemos cosas. Pero bueno, el comportamiento del algoritmo se corrige en medida a cuántas veces falló al momento de etiquetar los datos de prueba con respecto a las etiquetas que trae, y posteriormente modificar su comportamiento para las siguientes ejecuciones.

Aquí tenemos un pequeño diagrama. En nuestro diagrama hay una entrada de datos, en este caso son estas figuras, estas tres familias de figuras diferentes: un círculo, un triángulo naranja y un cuadrado morado, y están completamente desorganizados. Estos entran al algoritmo donde previamente el supervisor dio un conjunto entrenamientos y le dio las etiquetas que es la salida esperada para que el algoritmo sepa que va a clasificar tres cosas diferentes. El algoritmo previamente gracias a esas etiquetas sabe que va a clasificar cuadrados morados, triángulos naranjas y círculos verdes.

Entonces, el algoritmo al recibir estos datos de entrada simplemente hace el procesamiento necesario para clasificar en cada uno de esos tres grupos. ¿Cuál fue lo que tuvimos que haber hecho previamente? Pues nosotros delimitar el conjunto de entrada y delimitar las salidas esperadas, y posteriormente el algoritmo con el conocimiento que adquirió ya pudo trabajar con una serie de datos.

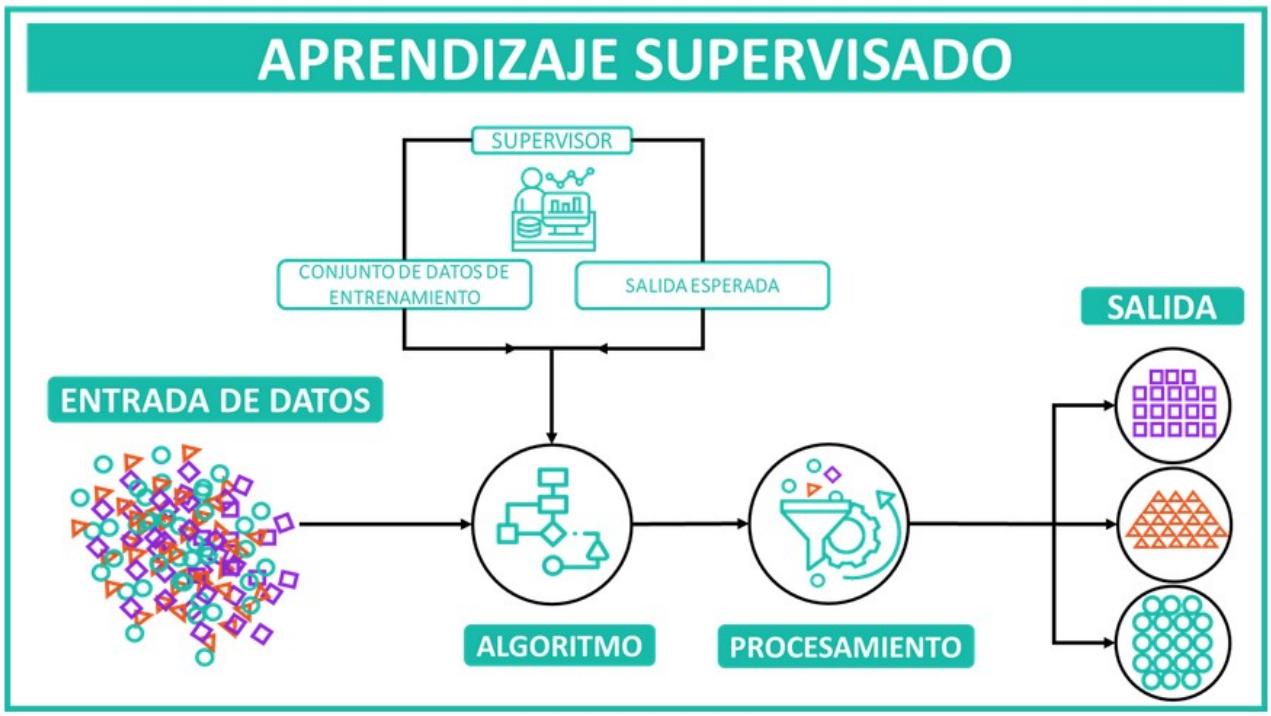


*Figura 7: Aprendizaje supervisado*

## Aprendizaje no supervisado

En el aprendizaje no supervisado ya no existen esos conjuntos de datos de etiquetados con los que antes podía entrenar y que después se buscaban extender el conocimiento áreas nuevas.

En este caso estos algoritmos suelen tener un comportamiento exploratorio es decir buscan características similares que puedan tener las entradas para clasificarlos. En este caso volvemos a tener los mismos tres objetos revueltos: volvemos a tener los cuadrados, los triángulos y los círculos, pero ahora la diferencia es que el algoritmo no tiene la información, la entrada es desconocida y no tuvo un conjunto de datos de entrenamiento.

Bueno, ¿cómo va a funcionar ahora? Ahora nuestro algoritmo puede interpretar o buscar relaciones muy diferentes en los datos. La primera podría ser "ok, tengo identificado unos objetos que tienen tres lados, unos objetos que tienen cuatro lados y unos objetos circulares", o podría decir "ok, tengo objetos naranjas, objetos verdes, objetos morados", o quizás haya otras, pero aún así el algoritmo intentaría separar nuestros datos en los conjuntos. La diferencia principal es que estos conjuntos no

les puede asignar una especie de etiqueta. No... no podría decir "ah... estos son cuadros morados, estos son triángulos naranjas, estos son círculos verdes", simplemente sabe que los tres son diferentes a partir de interpretaciones que encontró de los datos.

Este algoritmo se modificó a través de estar encontrando características. De ahí es de donde se dice que este tipo de algoritmos tienen un carácter exploratorio, buscan encontrar relaciones entre los datos.

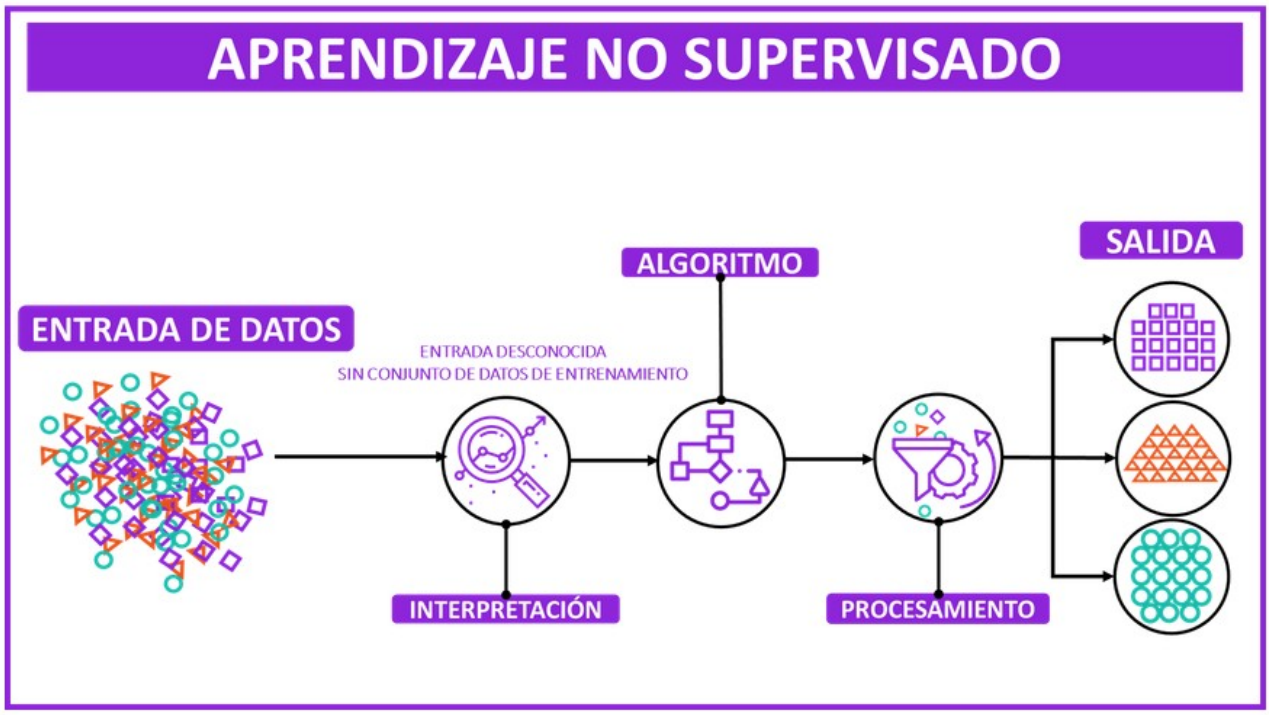


*Figura 8: Aprendizaje no supervisado*

## Aprendizaje reforzado

Las técnicas de aprendizaje reforzado son técnicas que intentan imitar el cómo los humanos y algunos animales aprenden del medio ambiente. ¿A qué nos referimos con esto? Cuando tú haces una acción, si esa acción es satisfactoria para ti, tú la empiezas a considerar una buena acción, pero si esa acción trae algún castigo hacia tu persona tratas de evitarla.

Tomar algo dulce generalmente es estimulante y te hace sentir feliz, entonces lo consideras algo bueno. Tocar algo hirviendo suele quemarnos y lastimarnos, y lo consideramos como algo malo.

De una idea similar es la que surge las técnicas de aprendizaje reforzado. ¿Cómo son las técnicas de aprendizaje reforzado? Las técnicas de aprendizaje reforzado: el algoritmo realiza una acción y esa acción puede traerle una recompensa o una penalización.

Si le genera una recompensa significa que la acción fue buena, entonces, en siguientes iteraciones el algoritmo va a tratar de imitar ese comportamiento bueno. Pero si una acción le produjo una penalización la va a tratar de evitar completamente.

A medida de cómo nosotros decidamos podemos eliminar esas acciones o simplemente darles... reducirles la probabilidad en la que se ejecuten. Pero bueno, en este caso nuestro algoritmo empieza a generar acciones y un agente le va a decir si esa fue una recompensa positiva o algo negativo. Eso

va a modificar el ambiente y el algoritmo va a estar estancado dando vueltas en estas iteraciones, decidiendo cuándo las acciones son buenas, cuándo son malas, y a partir de eso generará lo que se conoce como aprendizaje, empezará a generar reglas, modelos, empezar a aprender qué acciones evitar, qué acciones reproducir, hasta que logre, en este caso, volver a separar nuestras tres entradas, que son las mismas figuras que hemos estado jugando, en tres grupos diferentes.

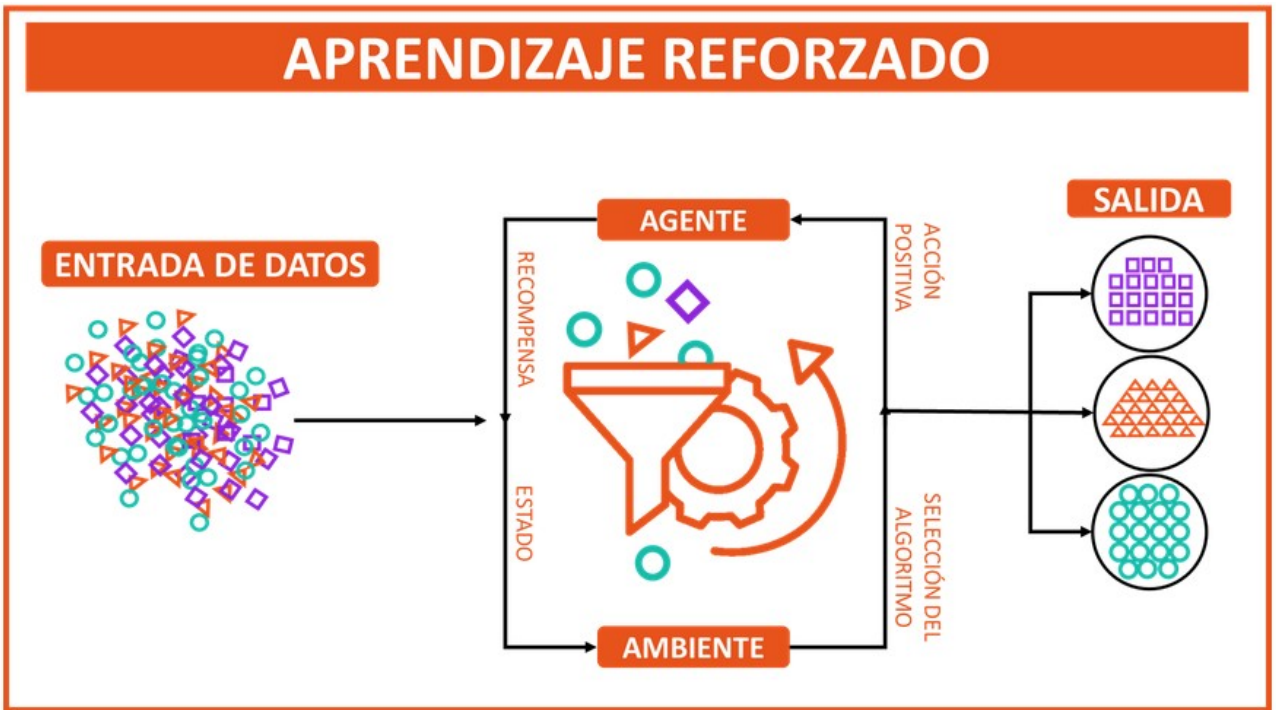


*Figura 9: Aprendizaje reforzado*

Estas técnicas, en resumidas cuentas, son cómo los programas están diseñados por dentro para que aprendan. No son las únicas de aprendizaje. Estas técnicas directamente no van a estar… no van a ser un algoritmo como tal, sino que estas ideas las vamos a introducir dentro de los algoritmos... de los algoritmos de machine learning como los de regresión, los de árboles de clasificación, entre otros, internamente.

Pero bueno, a medida que lo vayamos ocupando o que vayamos ahondando más te iremos recordando qué conceptos son necesarios que sepas.

En siguientes secciones hablaremos más a detalle de qué es el modelado de datos y empezaremos a trabajar con la librería scikit-learn.

## Scikit-learn

Ya vamos a empezar a trabajar con todo lo que es modelado de datos, pero para esto vamos a introducir una librería que vamos a estar ocupando en esta semana y en semanas posteriores para realizar este trabajo.

La librería en cuestión se llama scikit.learn y ahora preparamos un cuaderno en el que simplemente va a ser una introducción rapidísima de qué es scikit-learn y cómo funciona, o más bien, para qué funciona. Bueno, vamos a nuestra carpeta que tenemos de "cuadernos", en mi caso que está en el escritorio, en su caso donde la hayan descargado, y abramos el cuaderno de "scikit-learn".

Este cuaderno simplemente cuenta con una imagen, la imagen es tomada de la página principal de scikit-learn, y simplemente es para que lo leamos juntos y veamos de qué trata. Dice: scikit-learn es Machine Learning en Python, entonces es una librería que vamos a ocupar para toda la parte de machine learning: modelado, clasificación, regresión, clustering, a partir de ahora la vamos a trabajar y en el resto de este curso.

¿Qué dice? Dice que cuenta con herramientas simples y eficientes para minería de datos y análisis de data. También es accesible para todos y es reutilizable. Eso es muy genial porque pues...código sencillo lo podemos utilizar en diferentes cosas. Entonces, ¿qué más hace scikit-learn?. Scikit-learn dice aquí que está construido para ser trabajado en NumPy, SciPy y en matplotlib. Matplotlib es una librería que vamos a ocupar en este curso para trabajar con toda la parte visual.

Si no has tenido experiencia previa con esta paquetería no te preocupes. En el desarrollo de este curso vamos a ir tomando elementos aprendiendo a utilizarlos, viendo cómo puedes modificar unos parámetros para hacer más bellas tus gráficas o tus visualizaciones. Además es una paquetería gratuita, es opensource, lo cual hace que en internet haya muchísimos recursos para aprender o tutoriales o foros con gente que está interactuando contigo para que te puedan dar ideas.

Bueno, vamos a ver qué cosas nos puede ofrecer la página. La página nos ofrece algoritmos de clasificación, de regresión y de agrupamiento, o en inglés "clustering", y no sólo nos ofrece los algoritmos, también nos ofrece ejemplos. Entonces, si sientes en algún momento que hay un tema que no te queda el 100% claro en esta semana o en semanas posteriores te recomiendo mucho que busques algún ejemplo adicional, inténtalo, pero ten en cuenta que algunos de estos ejemplos a lo mejor rebasan un poco el nivel básico de programación.

Eso no debe espantarte porque, al menos en esta página, mucha de la documentación te va a ir guiando de qué hace cada uno de estos ejemplos. Pero bueno, esta paquetería la vamos a ir ocupando a partir de esta semana y por el resto del curso. Mi recomendación es, si tienes minutos adicionales y quieres aprender un poquito más, trata de leer un poquito de la documentación o de algún ejemplo que te parezca llamativo.

## Creando nuestro primer modelo

En esta sección vamos a empezar creando nuestro primer modelo de Data Science. Para ello vamos a utilizar un conjunto de datos llamado "Boston", que en realidad es los precios de casas en Estados Unidos, en la ciudad de Boston.

Esto ya viene incluido en scikit-learn. Lo podemos importar a través de "sklearn.datasets" y lo vamos a llamar "load_boston( )", luego lo mandamos a llamar como una función y lo vamos a guardar en esta variable llamada boston_dataset

Lo ejecutamos. Ok.

Este conjunto de datos contiene un atributo llamado "descripción". Si nosotros lo ejecutamos y lo imprimimos nos empieza a dar todo lo que contiene nuestro Data Set, es decir, una descripción. Este texto está un poco grande, entonces vamos a reducirlo a que... del carácter 148 a 1,225 para verlo ya resumido.

Entonces, tenemos que este conjunto de datos tiene trece atributos entre numéricos y categóricos, y nos dice cuáles son los atributos.

Algunos de ellos, por ejemplo, dice el crimen per cápita por ciudad, nos dice, por ejemplo, el precio en miles de las casas, nos dice el porcentaje de pobreza en la población, nos dice el número promedio de cuartos por por propiedad.

Entonces, toda esta información ya está contenida en esa base de datos. Vamos a leerla con Pandas para hacerlo un poco más agradable. Vamos a importar Pandas y a la base de datos le vamos a decir "dame los datos, dame los nombres de las columnas", y eso lo vamos a guardar en un data frame. Le vamos a dar ".head( )" para que nos de los primeros cinco registros, y nos lo muestra de esta manera.

Ahora, estas características son las descritas acá arriba. Nos falta una, nos falta el costo promedio de las casas. Esta es nuestra variable de respuesta. Nos falta agregarla y la podemos agregar de esta manera: le decimos "df" en la columna, como está escrito aquí, el precio medio va a ser igual al data set con su columna promedio, o columna target. Entonces, lo hacemos y ya nos acompletó la la tabla Pandas.

Podemos verificar la forma de esta base de datos con ".shape" y no va a decir que tiene 500 registros y 14 columnas que coincide con el número de columnas que manejaba acá arriba.

Ahora, el primer paso sería describir la información para empezar a entenderla.

Nos dice que todos tienen 506 registros por lo tanto no hay valores nulos, entonces ya ahí vamos en beneficio, ya no tenemos que limpiar eso. Nos dice los promedios, nos dice las desviaciones estándar, nos dicen los mínimos, el primer cuartil, segundo cuarto y el tercer cuartil, nos dice los valores máximos.

Vamos a graficarlo para darnos una idea de qué estamos haciendo, pero no vamos a graficar nada más así, lo que vamos a hacer es... vamos a decirle que grafique las correlaciones de esta tabla, es decir, nos va a decir las relaciones que tienen las columnas unas con otras, todos los pares de columnas.


*Figura 10: Mapa de calor*

Una vez creado nuestro mapa de calor con las correlaciones tenemos un montón de valores regados en la cuadrícula. Lo que representa esos números es qué tan correlacionados estos pares de variables son.

Por ejemplo: ZN y CRIM tienen un valor de -0.2, es decir, casi no están relacionadas. Ahora, los valores que nosotros estamos buscando aquí es en esta última columna.

Queremos valores o muy grandes o muy pequeños de correlación. Para recordar un poquito de la correlación un valor pequeño es -1 y un valor grande es 1, ese es el rango en el cual puede tomar la correlación.

De la lista que tenemos aquí, de la columna, los valores más extremos son este de aquí "RM" y "LSTAT". Los valores que tienen son 0.7 y -0.74, esto indica, recordando que RM es el número de cuartos por propiedad, esto nos indica que a medida que hay más cuartos en la propiedad el valor de la propiedad crece.

De forma contrario LSTAT, que es el porcentaje de pobreza en el área, nos indica que a mayor pobreza el valor de la casa decrementa. Por eso es una correlación negativa.

Una vez que ya identificamos algunas variables que nos pueden ayudar a hacer una predicción del valor de la casa vamos a usar de esta otra gráfica que se llama "pairplot". Esto nos va a graficar de manera similar al mapa de calor unas variables contra otras pero lo vamos hacer de una manera reducida ya solamente tomando las variables que nos interesan.

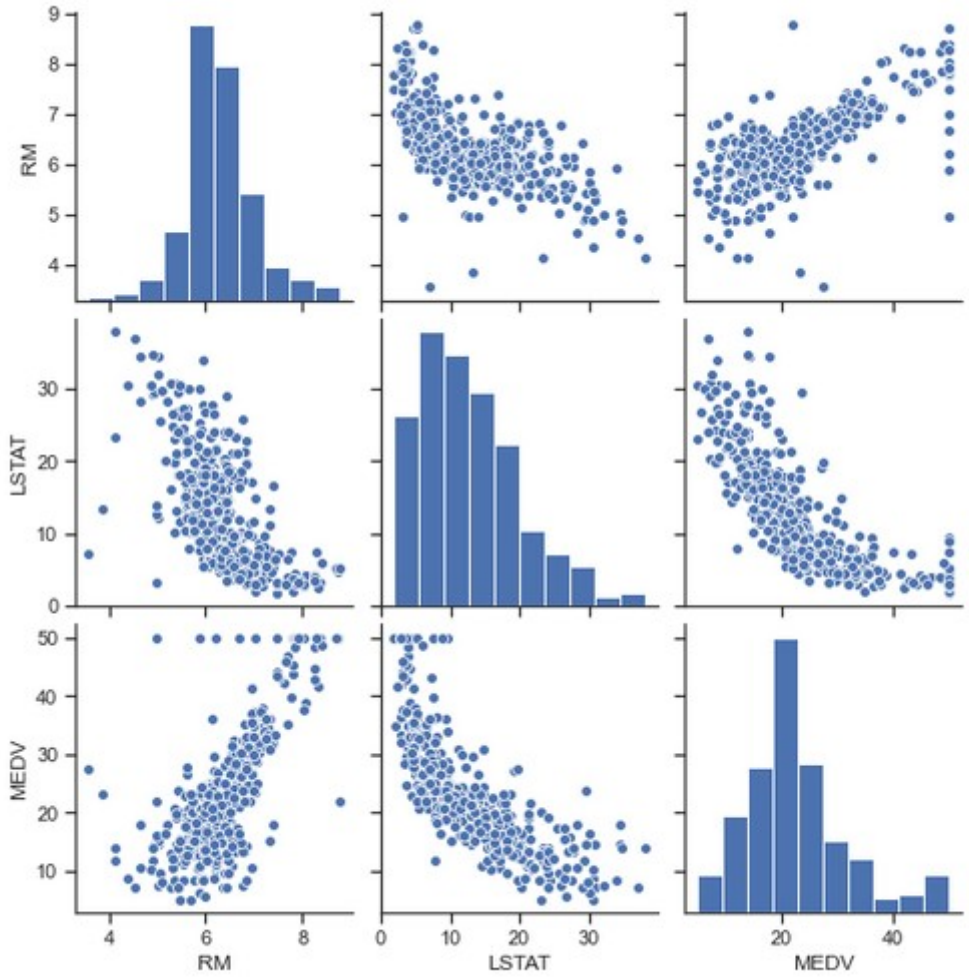


*Figura 11: Pairplot*

Tenemos... le pasamos el "df", que es nuestro data frame, y le pasamos nada más las variables RM, LSTAT y el valor de las casas. Cuando dos valores se encuentran en la diagonal nos muestra la distribución de los datos. Por ejemplo RM tiene una distribución alrededor de 6. 5 y 6, esa es su media. LSTAT, que es el porcentaje de pobreza, se agrupa alrededor de 10%, y el valor medio de las casas se agrupa alrededor de los 20, que recordamos que son millones, 20 millones.

En las demás celdas de esta gráfica nos muestra todos los puntos de nuestro data frame construidos de forma de que en el eje de las "y", en la segunda celda, tenemos el número de cuartos contra LSTAT.

¿En realidad cuáles gráficas nos interesan? Nos interesan cuando el valor de la casa es la variable de respuesta porque es lo que estamos buscando. Entonces nos interesan estas de aquí. El valor de las casas está en el eje del las "y", y en el eje de las "x" tenemos nuestras variables de la base de datos las cuales nos van a ayudar a predecir el valor de las casas.

Entonces, las conclusiones que habíamos hecho con la gráfica de calor aquí las estamos confirmando. Tenemos que a medida que va creciendo el número de cuartos en la propiedad el valor de la casa va creciendo.

Ahora, esto no siempre pasa. Hay valores que están fuera de este conjunto principal de datos, pero la tendencia nos indica que esa es la lógica, verdad, que esa es la realidad: más cuartos, más valor.

De forma contraria cuando hay más pobreza el valor de la casa decrementa.

Ok. Ya que una vez analizamos la información y podemos interpretar eso de los datos, pues ya podemos crear un modelo que haga esas predicciones, que a partir de un valor de número de cuartos a un valor de pobreza nos puede decir cuánto cuesta la casa, o una estimación del valor de la casa.

Para ello vamos a utilizar un conjunto reducido de datos porque lo vamos a ver de manera conceptual y muy sencilla. Entonces, tener poquitos datos nos puede ayudar a ilustrar mejor estas ideas.

Para ellos vamos a utilizar la función "sample" la cual no para decir: "obtén una muestra de tamaño 10". Este random state es un número que nos va a garantizar que los resultados que nos muestran nosotros sean los mismos que les da a ustedes. Es un control de la semilla aleatoria que determina los números aleatorios de la computadora. En realidad la computadora no puede generar aleatoriedad, es una máquina de bits de unos y ceros, por lo tanto tiene algunos mecanismos pseudo aleatorios y los podemos controlar a través de esa semilla.

Entonces, si todos ponemos esta misma semilla los números aleatorios que produzcan van a ser los mismos para nosotros y para ustedes.

Ok. Entonces, primero guardamos el conjunto original en "df_" para tenerlo reservado, y ahora sí, empezamos a modificar df. Para realizar un modelo de regresión vamos a utilizar el modelo más sencillo de todos que es una regresión lineal.

En él trata de ajustar o una línea al conjunto de datos que le presentemos, y es costumbre en el modelado utilizar "x" como variable independiente o variables independientes y "y" como variable dependiente o de respuesta.

En estos casos es RM, que es el número de cuartos, lo vamos a utilizar como "x", y "y" como el valor del costo de las casas. En particular hay que hacer una transformación en los datos.

Hay que darle este valor "reshape", o esta función "reshape", el cual les va a decir que el número de registros es indefinido, pero queremos una columna. Vamos a guardar esos datos como "x" y como "y". Ok.

Ahora. ¿Cómo podemos visualizar el modelo o cómo podemos hacernos a la idea que es un modelo? Lo podemos ver como que es una función, la función que por el momento es desconocida va a recibir una variable "x", que la que acabamos de crear, que son las variables independientes. Esa función nos va a producir una "y" que va a ser la predicción, pero esta "y" no es la original, no es la que nosotros conocemos como la verdad absoluta que es los costos de las casas reales.

Lo que nos va a producir es una estimación. Entonces no le podemos llamar "y" simplemente. Por lo general se utiliza una "y" testada que nos va a decir qué es la estimación. Ok. Entonces, ya tenemos estos elementos definidos:

"f" va a ser nuestro modelo, "x" va a ser nuestros datos de entrada y "y" va a ser la predicción, "y" testada va a ser la predicción.

Ahora sí, estamos listos para poder crear la predicción. Vamos a utilizar la regresión lineal y le vamos a decir "ajusta". ¿Ajusta con qué? Ajústate con los valores de "x" y con "y" que te estoy presentando, y esto nos va a generar un modelo que le vamos a llamar "reg". Ahora que ya tenemos "reg" y ya está ajustado le podemos decir "haz una predicción".

¿Con qué? Con los mismos valores de "x" para ver cuáles son las estimaciones que está generando y eso lo vamos a guardar como "y" testada o "y_hat" y vamos a imprimirlo.

Ok. Para los 10 valores de "x" nos produjo estas respuestas, pero pues los números así no nos dicen mucho, vamos a graficarlo. Primero vamos a hacer una gráfica de scatter donde en el eje "x" vamos a poner todas nuestras variables "x" y en "y" todas la variables "y".

Enseguida de eso vamos a crear otra gráfica en la cual vamos a utilizar en el eje "x" las mismas pero en la respuesta vamos a utilizar las predicciones, "y_hat", y vamos a marcarlo como una línea roja. Ok, ya lo tenemos. Ahora, cada uno de nuestros diez casas de ejemplo que tenemos están graficadas en estos puntos azules, la predicción, o nuestro modelo, está representado por la línea roja.

Verán que la línea roja no toca todos los puntos sino gen que hizo una generalización de los puntos y de la tendencia que sufren estos. Es decir, si nosotros hacemos una predicción donde el número de cuartos es 7 pues nos va a dar algo generalmente alrededor de los 28 millones. Vamos a confirmarlo. Vamos a hacer esa misma gráfica de scatter, vamos a volver a graficar nuestro modelo en la línea roja y vamos a ver unos puntos de prueba.

Tenemos los puntos 6 que nos va a predecir aquí... este 6. Tenemos el 6.5 que sabemos que no es posible porque no podemos dividir el número de cuartos como ejemplo. Y vamos a agregar otro punto, vamos a agregar el 7 que sería una propiedad con siete cuartos. Ok. Esto lo vamos a llamar "test_points_X".

Y ahora, con nuestro modelo, vamos a realizar la predicción de estos nuevos puntos. Observen la estructura. Es un arreglo de arreglos pequeños. De eso tiene que ser la estructura. El resultado de esa predicción tiene que ir guardado en "test_points_Y", así les nombramos nosotros, y ahora vamos a volver a hacer un scatter, pero ahora ya no va a graficar los puntos originales, va a graficar los puntos que le estamos mandando en "x", y en "y" va a ser la predicción de nuestro modelo.

Ahí está. Entonces, nuestro modelo dice que para el 6...el 6 va a estar sobre la línea roja en este punto que es alrededor de un 16... 17...17 millones. El 6.5, que no estaba nuestro conjunto de datos original, lo predice alrededor de 24. Y para el 7 que tampoco estaba en nuestros datos originales, lo está preduiciendo en un 28, alrededor del 28.

Entonces ya tenemos un modelo que está realizando predicciones de datos que nunca antes había visto, y ese modelo lo generó a partir de la colección de datos que teníamos. Entonces, de esa forma funcionan los modelos.

En la siguiente sección vamos a ver cómo podemos evaluar un modelo y decir qué tan bien o qué tan mal está funcionando.

# Métricas de error

En esta sección vamos a evaluar el modelo que creamos en el video anterior. Para ello vamos a seguir utilizando el mismo archivo que hemos usado que es el cuaderno "creando nuestro primer modelo". Si ven, aquí arribita está lo que hicimos en el video anterior y vamos a continuar a partir de ahí. Bueno, ahora la pregunta es: una vez que creamos nuestro modelo ¿cómo lo podemos evaluar?

Bueno, para ello hay algunas funciones llamadas métricas de error. Estas nos va a arrojar un aproximado de qué tan bien o qué tan mal está desempeñándose nuestro modelo.

Para verlos más detalladamente podemos utilizar este enlace de aquí, que si lo abrimos nos va a decir todas las métricas disponibles en sklearn. Estas varían dependiendo qué tipo de problema estamos utilizando. Por ejemplo, hay métricas para clasificación, métricas para regresión y métricas para clustering.

De ellas nosotros, para una regresión lineal, vamos a utilizar el "r" cuadrada, o el coeficiente de terminación. Este mide una manera general qué tan bien mide nuestro modelo la variable independiente. Esta métrica puede tomar un valor de 1 si el ajuste es perfecto, que raramente va a suceder. Va a tomar un valor de 0 si es equivalente a solamente tomar el valor promedio, va a tomar un valor negativo si el modelo en sí es muy muy malo.

Los valores aceptados de "r" cuadrada comúnmente rondan entre 0.7 y 0.9 para decir que el modelo es bueno o es aceptable. Entonces, vamos a utilizarla.

Para importarla solamente utilizamos "sklearn.metrics". Esto nos va a importar las métricas de error, y vamos a importar esta especifica: la "r" cuadrada.

La mandamos a llamar de esta forma, r2 y vamos a pasar la "y", que son los datos verdaderos originales, y la estimación, o la predicción. Y esto nos va a dar un valor numérico: es de 0.61. 0.61 nos indica que la predicción es... no tan buena pero pues puede llegar a ser aceptable.

Pero, ¿podemos mejorarla? Sí, evidentemente sí lo podemos mejorar. Podemos hacerlo tratando de cambiar la variable, por ejemplo... acá usamos en el modelo... usamos el número de cuartos como variable de predicción. Vamos a cambiarlo por el nivel de pobreza. Vamos a repetir los mismos pasos, nada más que ahora vamos a cambiar el valor de la "x". Entonces, hacemos eso... hacemos el ajuste de la regresión lineal y luego hacemos la predicción y lo guardamos como "y_hat".

Ahora, vamos a volver a imprimir la gráfica. Hacemos un scatter con "x" y "y", hacemos la gráfica que representa nuestro modelo y vamos a utilizar dos datos de prueba: el 8 y el 11. Eso... vamos a hacer una predicción, y vamos a verificar estos nuevos puntos. Ok. Entonces, primero nos dice que para el 8 esto representa un valor promedio de las casas de 26 millones, para un valor de 11 representa un valor de costo de 21 millones, es decir, a medida de que incrementa el porcentaje de pobreza, porque es la variable que estamos analizando, el valor de la casa va a decrecer, y lo podemos ver en nuestra gráfica: a medida que incrementa el nivel de pobreza decrementa el valor de la casa.

¿Pero esto mejora o empeora nuestra regresión? ¿Cómo se comporta comparado con el modelo anterior? Ahora, si lo medimos nos da un 0.73, es decir, la nueva selección de nuestras variables mejoró nuestra estimación de la variable de respuesta: era 0.61 y subió hasta 0.73.

Entonces, ya es más aceptable. ¿Pero aún es posible mejorarlo? Sí. Lo que vamos a hacer es utilizar una regresión polinomial. En esta sección vamos a dejarlo un poquito como caja negra... es algo ya un poquito más avanzado, que vamos a ver en secciones posteriores, pero lo vamos a ver de forma ilustrativa, este código lo que hace es hacer una regresión polinomial y la va a graficar para nosotros. Entonces, lo hacemos, lo ejecutamos y nos da esta cosita de aquí, que este nos va a decir el grado del polinomio el cual estamos utilizando como modelo.

Entonces, si lo movemos de 1 a 2 cambia de una regresión lineal a una cuadrática, luego a una cúbica, y etcétera, dependiendo el grado del polinomio.

Lo que tenemos aquí es el valor de respuesta de r2 con una regresión lineal simple. Al hacerla cuadrada es el valor de la regresión cuadrada. Pero el modelo es muy similar, por lo tanto no hay cambios grandes. Al hacerla cúbica ya es donde empieza a cambiar un poco más esta métrica de error, ya es de 0.80; al hacerla 4: 0.81; 5 es 0.90. Entonces la lógica nos diría hasta el momento que este valor de grado lo subamos hasta 10.

Y nos da un valor del 0.98. Entonces, pues podríamos decir que es muy muy bueno. Veamos nuestra gráfica. Lo que está pasando es que el modelo está tocando todos los puntos del conjunto de datos. Pero qué pasaría si le damos un valor nuevo. Supongamos que queremos predecir un valor de pobreza del 5%, lo que nos va a dar es un valor que está hasta acá abajo y nos está diciendo que la casa tiene un valor de -887,000,000, es algo incoherente, no puede ser eso.

Nosotros al subir la complejidad de nuestro modelo estamos reduciendo por mucho su capacidad de predicción. Entonces, un grado de complejidad alto de nuestro modelo no nos funciona. Si lo decrementamos, pues visualmente podemos ver que va mejorando, se adapta mejor a los datos de una manera general. Pero cuál es el punto de donde nos detenemos. Sólo tenemos una referencia visual. Nuestro r2 no nos está funcionando para nada. ¿Cómo podríamos hacer eso? O sea, automatizar la forma en la que determinamos si un modelo es bueno o no. O, ¿cómo podemos escoger el modelo ideal? Bueno, vamos a llevarnos a los extremos. Tenemos la primera parte que es un modelo muy simplificado y la segunda que es un modelo muy complejo.

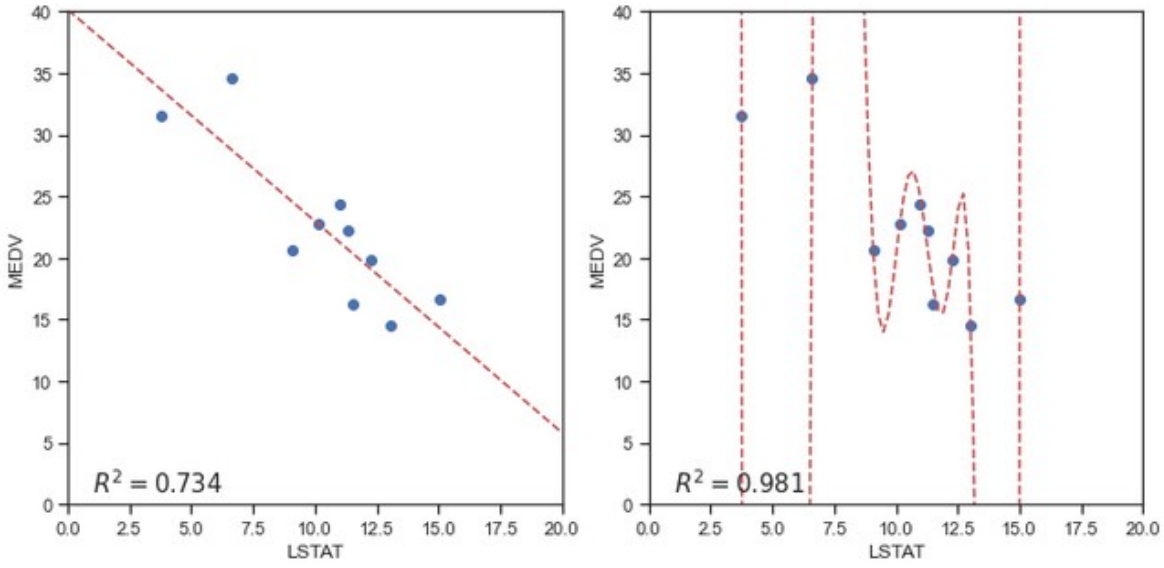


*Figura 12: 2 modelos con grados diferentes*

El modelo de la izquierda es una línea, por lo tanto, por la naturaleza de datos, nunca va a poder tocarlos todos. Entonces, ajustarse a los datos es una tarea imposible para la línea.

Esto se dice que tiene un ajuste muy abajo de la perfección, es decir, es un underfit, por el inglés. Otra forma de decir esto es que el modelo tiene un alto sesgo, o un alto bias. En el segundo caso el modelo es muy muy complejo. Se adapta tanto a los datos que ya perdió la generalidad el conjunto de datos. Aprendió tanto de los puntos la relación que había, tanto de los posibles errores que hubo a la hora de recopilar los datos, o incluso del ruido.

Se dice que este modelo hizo un overfit, o sobre ajustó a los datos, de forma que hacer una predicción confiable es imposible. Eso también se dice que el modelo tiene una alta variabilidad, o una varianza alta.

Entonces, ¿qué es lo que buscamos? ¿El de la izquierda o el de la derecha? En realidad lo que buscamos es un punto intermedio donde el modelo tiene cierta flexibilidad pero no tanta como para sobre ajustarse a los datos. Entonces, el reto para un data scientist es encontrar el balance entre un underfit y un overfit, lo que se podría considerar un buen modelo, algo que represente la generalidad de los datos y no tan general ni tan específico.

Bueno, pero entonces, ¿cómo podemos predecir a través de nuestras métricas de error cuál es un buen modelo? Para ello se utilizan splits, lo cual divide nuestros datos en dos: uno de pruebas y otro de entrenamiento.

## Entrenamiento y prueba

En la sección anterior vimos cómo un modelo puede llegar a tener un bajo ajuste: un underfit; o un sobreajuste un overfit. Una manera de contrarrestar el overfit o el underfit es haciendo un split de los datos. Eso significa que vamos a dividir la información en dos conjuntos: uno de entrenamiento y otro de prueba.

Esto lo podemos ver en nuestro cuaderno de Jupyter "creando nuestro primer modelo". Es el mismo que hemos estado utilizando. Vamos a describir qué es cada uno de estos conjuntos y para qué funcionan. El conjunto de entrenamiento es el cual se va a utilizar sola y exclusivamente para entrenar el modelo. No se va a utilizar para otra cosa.

El conjunto de prueba solo se va a utilizar para validar o para probar qué tan bien está funcionando nuestro modelo. La separación es debido a que si tenemos muy buen resultado en entrenamiento con validación, o sea, datos que nunca ha visto el modelo, podemos chequear qué tan bien está comportándose en la generalidad para poder combatir efectivamente el over training.

Entonces, cuando nosotros entrenemos nuestro modelo, ya cuando lo vayamos a evaluar, lo vamos a evaluar dos veces: uno con el conjunto de entrenamiento y otro con el de prueba.

Un error pequeño en entrenamiento pero uno alto en prueba nos indica que el modelo está sobre entrenado. Entonces hay que encontrar un balance en qué tan bien queremos que salga nuestro modelo entre entrenamiento y prueba.

Primero hay que identificar que el conjunto de pruebas sea lo suficientemente grande. Por lo general se utiliza el 80% de los datos como entrenamiento y el 20% de prueba. Es una regla de dedo pero se puede ajustar a las necesidades que tengamos.

También hay que ser muy cuidadoso a la hora de usar el entrenamiento. Tenemos que irlo rotando de vez en cuando porque si no le estamos induciendo aprendizaje de forma indirecta al modelo. Por lo tanto tenemos que moverlo, rotarlo para no estar constantemente usando el mismo conjunto de prueba.

En esta gráfica podemos ver más o menos lo que está pasando a la hora de utilizar estos modelos.

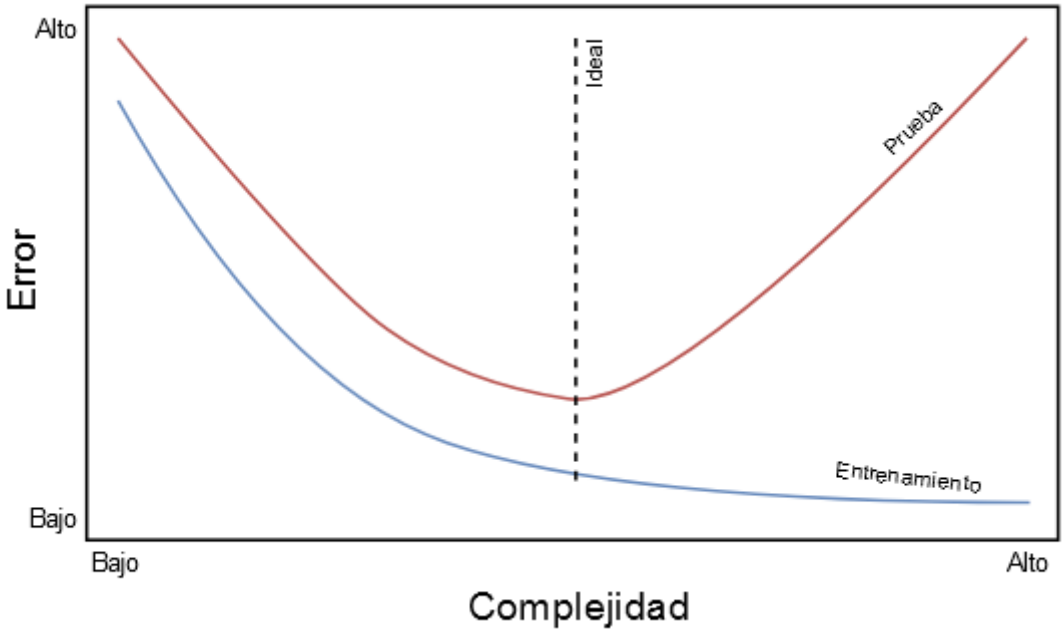


*Figura 13: Gráfico de complejidad de un modelo versus su error*

El eje de las "x" nos indica qué tan complejo es el modelo que estamos armando: bajo sería un modelo lineal y muy alto sería un modelo muy muy complejo, ya sea uno de un polinomio muy alto. En la parte de las "y" tenemos el error, ya sea alto hacia abajo. Al principio, o con modelos lineales, generalmente el error es alto para ambos casos: para el conjunto de prueba y para el conjunto entrenamiento delimitados por la línea roja y la línea azul.

A medida que la complejidad incrementa el error tiende a disminuir hasta un punto en el que, si seguimos incrementando complejidad, el error de prueba se empieza a disparar y sale muy contraproducente.

Nuestra tarea es identificar el punto ideal donde podemos hacer detener la complejidad para poder así obtener el mejor de los dos mundos, el mejor... la prueba con un entrenamiento bajo.

Vamos a usar un split de los datos, para ello vamos a recuperar nuestro conjunto de datos original y vamos a hacerle una muestra de 20 valores. Lo hacemos e importamos el "train_test_split", esta es una herramienta que ya viene incluida en scikit-learn que nos va a permitir hacer la división de los datos de una manera intuitiva y sencilla. Entonces la mandamos a llamar así: train_test_split, y le vamos a pasar nuestro conjunto "x" nuestro conjunto "y", la cantidad de datos que queremos dividir para prueba y una semilla aleatoria, y lo que nos va a devolver son cuatro datos o cuatro variables: es la "x" en entrenamiento, la "x" en prueba, la "y" en entrenamiento y la "y" en prueba.

Lo ejecutamos y los podemos imprimir. Vamos a imprimir sus formas. La forma del conjunto original era 20 registros porque hicimos un muestreo con 14 columnas. La "x" original que tenemos

aquí ya es seleccionando nada más la variable LSTAT, por lo tanto son 20 registros y una columna. A la hora de dividirlo en entrenamiento y prueba lo dividió donde hay 13 registros y una columna y 7 registros y una columna, automáticamente hizo la separación de los datos.

Ahora vamos a visualizar lo que está pasando cuando incrementamos la complejidad de nuestro modelo. Ahora, tenemos algo muy similar a lo que teníamos en el vídeo pasado pero ahora nos reporta dos tipos de error: el "r" cuadrada de entrenamiento y el "r" cuadrada de prueba. Lo que pasa al incrementar la complejidad... veamos que inmediatamente ambos errores mejoran. Recordemos que la "r" cuadrada 1 es bueno y 0 es malo.

Cuando incrementamos a 3 ambos errores siguen creciendo, entonces van en positivo. Si lo vemos en la curva esta siguen decrementando el error, por lo tanto es bueno, ya nos estamos aproximando nuestro punto ideal. Cuando pasamos a un grado 4 el "r" cuadrada en prueba empieza a empeorar: pasó de 68 a 54, sin embargo el entrenamiento sigue mejorando: pasó de 68 a 70. Entonces aquí ya empezamos a ver cómo el error empieza a subir en prueba y el entrenamiento sigue bajando. Entonces, ya pasamos nuestro punto ideal.

Si seguimos incrementando esto se hace todavía más evidente, entrenamiento sigue en 70... sigue bajando en 70, y "r" cuadrada en prueba sigue empeorando.

Esto va a seguir siendo hasta que ya llega un punto en el que de plano ya erradico el error, verdad. En "r" cuadrada de prueba ya tiene un -100. Recordamos que el negativo es malo, ya para que llegue a un -100 ya es muy muy malo. Lo llevamos a 7, sigue empeorando; a 8, sigue empeorando y así va a seguir por el resto. Sin embargo, en entrenamiento tenemos un muy buen valor, tenemos de 0.93.

Entonces, ¿cómo podemos encontrar ese balance? Bueno, es algo que vamos a estar viendo en secciones posteriores, pero quiero que quede muy claro la idea de que hay un punto en el que si seguimos incrementando la complejidad nuestro modelo va a ser muy malo a predicciones futuras. A parte de hacer un split para poder identificar el punto perfecto para hacer el corte perfecto de complejidad existe otro mecanismo para poder contrarrestar el over training que se llama cross validation.

Es similar al test nada más que el test lo hará rotando. Va rotando cuál es la parte de prueba y cuál es la parte de entrenamiento, de forma que construye un modelo que sea general para cada uno de esos conjuntos de prueba.

## Cross Validation

En la sección anterior estuvimos hablando del split que podemos realizar entre entrenamiento y prueba para, de esa manera, utilizar la parte de validación para poder encontrar qué tan buenos son nuestros modelos. Una desventaja de usar este modelo es que cuando, una vez que utilizamos el modelo de pruebas, prácticamente ya no lo volvemos a utilizar, no le damos un segundo uso, entonces, estamos desperdiciando muchos datos, o al menos tentativamente eso sería lo que estamos haciendo.

Una alternativa de ello es utilizar cross validation. Cross- validation es una técnica la cual nos va a permitir no dejar un tramo fijo de validación sino irlo rotando. En el cuaderno del video pasado que estábamos trabajando que se llama "creando nuestro propio modelo" tenemos la descripción de cross-validation. Entonces, vamos a él.

Cross-validation, dice, es una alternativa al método de división de datos, o el split. Es llamado validación cruzada o como comúnmente se conoce como "CV"... cross-validation. Con esta técnica, en lugar de generar un solo split, como les mencionaba, se generan múltiples cortes pero con la diferencia que es en tiempos diferidos.

Una manera sencilla de imaginarlo es que estamos haciendo un split de entrenamiento y prueba, acabamos con él, rotamos o movemos y lo volvemos a intentar. Ahora, ¿cuántas veces vamos a hacer este proceso? Bueno, aquí hay una variable determinada "k". El número de "k" es el número de divisiones o número de repeticiones que vamos a hacer de nuestro conjunto de datos total. Cada una de estas particiones lo que genera realmente es un conjunto de entrenamiento, un grupo de validación y un modelo entrenado.

Imaginemos el caso de que tengamos cuatro "k"s, así como lo tenemos en nuestro cuaderno, para cada uno de estos splits supongamos que tenemos un cross validation de tamaño 4 y supongamos que el primer modelo que vamos a generar es el que tiene las tres primeras partes como entrenamiento y la última de validación. Entonces, el modelo que generemos lo vamos a entrenar con esas tres partes, vamos a calcular como lo hicimos en el test validation, y vamos a calcular el error con la última parte. Y ese valor lo vamos a guardar, cuánto nos dio de error, ya sea con el r^2 o con alguna otra métrica para cálculo de error.

Una vez que acabamos con eso, en la sección de error o de validación, se mueve. Una vez que hay un movimiento volvemos a generar con el conjunto entrenamiento un modelo y que lo validamos con un conjunto de pruebas. Eso lo repetimos cuatro veces porque tenemos una k=4, y con eso ya tenemos los resultados de nuestro modelo.

Ahora falta sacar el promedio de esos valores. Y esto lo hacemos de la siguiente forma: el error reportado por cross validation es el promedio de los "k" ciclos. ¿Por qué? Porque son cuatro partes, cuatro modelos, entonces el error promedio de esos cuatro modelos es el error de cross validation, y utilizamos simplemente la fórmula de promedio, tomamos el error 1, 2, 3, 4, por los cuatro modelos, y los dividimos sobre k, y eso nos da el error general de nuestro sistema de predicción.

¿Qué ventaja tiene esto sobre el split normal? Bueno, la ventaja es que, como les mencionaba, podemos reutilizar los datos que de otra manera estuviéramos desperdiciando nada más utilizando una vez, y estar construyendo diferentes modelos los cuales podemos evaluar de forma general para obtener un error que se acercaría más a lo que sería ya cuando lo pongamos en producción.

Cross validation tiene la particularidad de que es un cálculo un poco más pesimista que el train_test, sin embargo, esto nos da una mejor idea de cómo se comportaría más adelante. Ahora vamos a hacerlo en Python.

No tenemos que estar haciendo todo nosotros, estarle diciendo cómo dividir los datos. Afortunadamente Python ya tiene un método el cual nos va a hacer las divisiones que nosotros esperamos, y esto lo hace a través de esta función, KFold.

KFold recibe tres parámetros: el número de splits que vamos a hacer, el número de divisiones, si queremos que los revuelva, los datos, antes de empezar a hacer los splits, es decir, si tenemos datos ordenados del 1 al 100 no va... el primer split no va a ser 1, 2, 3, 4, 5, 6, sino, lo va a revolver y va a seleccionar elementos aleatorios. Y en el caso de que queramos revolverlos nos pide una semilla aleatoria. En este caso como es algo demostrativo vamos a dejarlo que no haga el reordenamiento,

vamos a dejarlo que esté fijo y vamos a decirle que tiene un k=4, hay que preparar nuestros datos para poder hacer el cross validation.

Entonces, vamos a pasarle dos variables: 1, una variable independiente que es LSTAT, que... recordemos que era el índice de pobreza, y el valor esperado que era el costo medio de los hogares. Vamos a pasárselo como df y lo guardamos en data. Para hacer este cross validation hay una estructura muy específica que es un ciclo "for". Le vamos a decir: para cada train y test en todo nuestro cross validation va a empezar a iterar, y cada iteración va a ir imprimiendo en este caso específico los train y test que estamos esperando.

Entonces, vamos a hacerlo. Entonces, nos está diciendo que en el primer ciclo, en la primera iteración, es decir, en el primer movimiento del conjunto de prueba, el conjunto de prueba es 0, 1, 2, 3 y 4, que son los primeros 5 elementos y el resto lo voy a utilizar como entrenamiento. Una vez que terminemos con eso los va a rotar, va a tomar los siguientes 5 y el resto lo usa para entrenamiento, luego lo va a volver a rotar. Fíjense que el tamaño de pruebas acá pasó de ser de 5 a 6, se pasa a los siguientes 5, luego toma los siguientes 5 que son del 10 al 14, y luego los siguientes 5 que son del 15 al 19. Ok, esos van a ser nuestras cuatro iteraciones.

Ahora vamos a empezar a entrenar modelos porque ahora lo único que hicimos fue tomar los índices... los índices... fíjense que estos son puros índices, y los imprimimos. Ahora vamos a usarlos ya para empezar a predecir.

Entonces, cómo lo hacemos. Vamos a utilizar otra vez el kfold, que es el que nos va a dar el cross validation. Ahora lo vamos a poner en el mismo ciclo y vamos a empezar a preparar los datos. Como dijimos esta función lo que no está dando son los índices. Entonces, de los datos originales vamos a utilizar el "iloc", que "i" se refiere a la posición numérica. Y le vamos a decir: danos todos los datos de entrenamiento y de eso nada más darnos la columna LSTAT.

Recuerden que tenemos que utilizar este... esta parte de aquí que es values.reshape que le estamos haciendo una transformación de los datos diciéndole que el número de filas es desconocida, o indefinida, pero que queremos una columna. De forma similar lo vamos a hacer con la parte de variable de respuesta, que son los... el costo de las casas para entrenamiento, una vez hecho eso hacemos lo mismo tal y como está pero con el conjunto de prueba.

En este punto ya tenemos nuestros datos listos, entonces ya podemos hacer una regresión. Vamos a decirle: regresión lineal, entrena con el conjunto entrenamiento, y eso nos da un modelo que le vamos a poner "reg".

Con ese reg vamos a hacer la predicción. Vamos a predecir con entrenamiento y vamos a predecir con prueba, ¿para qué? Para obtener los valores del número de error e imprimirlo en pantalla. Vamos a calcular los errores con esto que acabamos de crear. Le estamos diciendo el y_train, que son los datos originales, y la predicción, todo esto en entrenamiento y lo mismo en validación y lo vamos a guardar en estas variables: r_train y r_test.

Ahora, lo vamos a imprimir y, para finalizar, lo vamos a ir acumulando en una variable externa que es esta avg, ¿para qué? Para después dividirlos entre 4 y obtener el error promedio. Entonces, lo ejecutamos… y aquí está, nos dio que para el primer k nos dio un error de casi 0.6 en r^2 en entrenamiento, y de prueba nos dio 0.593, es muy similar, en el segundo caso nos dio 5.89 para entrenamiento y 5.12 para prueba. De forma similar nos dio los otros dos casos. Ahora, estos en resultados individuales no nos interesan tanto. Lo que nos interesa es el acumulado, o más bien el

promedio de esos datos. Entonces, lo podemos dividir y nos da un 0.61 en entrenamiento y 0.53 en prueba.

Esto es un resultado esperado, que el entrenamiento sea un poquito más arriba que la validación. Y este es un caso más real o más cercano a la realidad que lo que nos daría un split normal. Pero pues nos falta verlo de alguna manera.

Entonces, esta celda de aquí, estoy código es lo mismo que el de arriba, solamente le agregamos las gráficas. Vamos a verificar cada una de las predicciones en el espacio de incógnitas para ello vamos a utilizar esta parte de aquí que es el "linspace" que lo que nos va a producir es 100 números distribuidos entre 0 y 20.

Vamos a utilizar... aprovechar nuestro modelo que ya está construido para hacer una predicción con esos puntos de forma que las "y" sean las variables de respuesta y nos pueda generar la línea. Vamos a hacer un scatter con el entrenamiento y un scatter con la prueba. Lo hacemos en formas independientes para que primero nos grafique puntos azules para entrenamiento y puntos naranjas para la prueba. Vamos a graficar nuestro modelo lineal y lo mostramos en pantalla con la función show. Entonces, lo hacemos y nos va a imprimir cuatro gráficas.

Cada una de éstas con su respectivo modelo. Entonces, aquí está. Tenemos en el primer caso dos puntos azules que son los que utilizó el entrenamiento y todos los naranjas son los que utiliza de validación. En el segundo caso se ve como rotan, ya no son estos mismos sino que ahora estos fueron utilizados como validación y estos de acá ya no son de entrenamiento sino, son de validación. Lo mismo pasa en los otros dos casos.

Aunque el modelo sea muy parecido, en realidad cambia ligeramente. Esta línea roja no es exactamente igual a ésta, ni igual a ésta, ni igual a ésta. Todos son diferentes. ¿Cómo lo sé? Pues simplemente porque los errores son diferentes. Sé que está construyendo modelos diferentes. Entonces, cross validation, para lo que nos va a servir no es para generar en realidad un modelo específico. Lo que nos va a ayudar más es para configurar nuestros parámetros de los modelos para poder, ahora sí, crear un modelo específico.

Cross validation nos ayuda más para darnos una idea de cómo funcionaría nuestro modelo en el mundo real.

# Semana 2

## La Regresión y su teoría

En esta sección vamos a empezar a revisar la teoría en conjunto de qué es una regresión lineal. Preparamos una libreta de Python que incluye la mitad teórica, que es la que vamos a revisar ahora, y más adelante revisaremos la parte práctica en la que no sólo veremos lo que es la regresión lineal, sino veremos un tema muy relacionado que es la regresión múltiple.

Para eso va a ser la misma libreta que vamos a estar trabajando. Entonces, vamos a la libreta que se llama "regresión lineal". Entonces, vamos a abrirla.

En esta libreta, antes de empezar a leerla en conjunto, aquí está un ejemplo para que vayamos leyendo y analizando, y ya después está la parte de regresión lineal y regresión múltiple. Ahora sí, vamos a empezar en orden desde el principio.

¿Qué es la regresión lineal? Bueno, la regresión lineal es un modelo que nos permite encontrar la relación que existe entre las variables dependientes, generalmente con el nombre de la variable "x", y la variable independiente, usualmente ocupamos la letra "y" para asignar a la variable independiente, que nos sirve para predecir el valor esperado de una variable cuando "x" toma un valor específico.

Este método implica suponer que existe alguna función de la forma de una línea recta, es decir, la ecuación y=mx+b, donde "m" es la pendiente que nos dice que tan inclinada está nuestra recta, y "b" es la ordenada en el origen, es decir, el punto "y" en el que se intercepta con el eje "y". Nos dice que existe una recta que es la representación más adecuada que más se asemeja a la distribución de los datos en un diagrama de dispersión.

En el mundo ideal todos los puntos van a coincidir con nuestra recta, sin embargo en la realidad no es así. Típicamente los diagramas de dispersión que tienen forma de línea recta permite que se ponga una línea supuesta que se asemeje mucho a los valores, pero no necesariamente tocará a todos. Más adelante en el ejemplo espero que esto tenga un poquito más de entendimiento.

Pero bueno, vamos a empezar en el ejemplo. Nuestro ejemplo empieza aquí. Tenemos una lista de 20 puntos o de 20 elementos tomados de una base de datos en la que hay dos variables: la variable "x" y la variable "y". Si nosotros graficamos estos 20 pares ordenados de coordenadas nos da este diagrama de dispersión, y a primera vista se ve que hay una tendencia a una distribución en forma de una línea recta.

¿Cuál es esa línea recta? No lo sé. La visualización nos demuestra que existe una línea recta que al menos pasará por donde la tendencia de los datos da. Pero bueno, para empezar con esto, ¿cuál es el procedimiento qué haríamos si tuviéramos que hacer manualmente esta regresión? Esta regresión se calcula utilizando la suma del producto de la "x" por la "y", la suma del valor de "x", la suma del valor "y", la suma de x^2 y de y^2.

Yo sé que parece que es demasiado trabajo, pero no se preocupen, Python cuando ya lo esté haciendo lo va a hacer por ustedes. Por el momento simplemente les quiero ejemplificar cómo hubiera sido el trabajo previo a usar una librería especializada. Entonces, para poder encontrar esa sumatoria que yo les comento. Necesitamos hacer una tablita y sumar los valores. En la primer columna tengo los valores de "x" que sumados dan 265.75. La segunda columna tengo los valores de "y" que sumados dan 401.1. Tercera columna tengo la multiplicación del "x" por el"y"... multiplicación de "x" por "y", y al final, después de multiplicar cada par, la suma... es ese 4800. En la cuarta columna tengo las "x" cuadradas que suman 4119 y la quinta "y^2". Entonces, con estos elementos que ya calculamos vamos a pasar a nuestras ecuaciones.

Las ecuaciones que vamos a manejar para poder encontrar la pendiente y la ordenada en el origen involucran calcular previamente la varianza y el promedio de cada una de estas variables. Bueno, para los que no recuerdan cómo es la fórmula del promedio, la fórmula promedio simplemente es sumarlo... todos los elementos y dividirlo entre el total de elementos. En este caso nuestra suma, que es este 265.75, lo dividimos entre 20 y nos dio este 13.287 de la misma forma que calculamos la de la "y", y la varianza, aquí es importante para la gente que no está familiarizada con estadística, la varianza existe realmente como dos tipos: la varianza de una población y la varianza de una muestra.

En este caso tenemos 20 elementos. Entonces, la varianza que vamos a estar ocupando en la fórmula que está presentada en su libreta es la varianza de una muestra. Por eso está esa aclaración.

La varianza en "x" de la muestra se designa con este símbolo y es esta fórmula la sumatoria de los valores al cuadrado, la que ya tenemos aquí arriba: 4119, menos n veces la multiplicación de la "x" promedio al cuadrado entre n menos 1 y nos da este valor y este valor.

Con estos valores ahora sí ya puedo construir mi pendiente y mi ordenada.

| | | |
|---|---|---|
| Promedio $\bar{x}$ | $\bar{x} = \sum \frac{x_1 + x_2 + \cdots + x_n}{n}$ | $\bar{x} = 13.287$ |
| Promedio $\bar{y}$ | $\bar{y} = \sum \frac{y_1 + y_2 + \cdots + y_n}{n}$ | $\bar{y} = 20.055$ |
| Varianza **x** (muestra) $\sigma_x^2$ | $\sigma_x^2 = \frac{\sum x_i^2 - n(\bar{x})^2}{n-1}$ | $\sigma_x^2 = 30.946$ |
| Varianza **y** (muestra) $\sigma_y^2$ | $\sigma_y^2 = \frac{\sum y_i^2 - n(\bar{y})^2}{n-1}$ | $\sigma_y^2 = 28.752$ |
| Pendiente $m$ | $m = \frac{\sum x_i y_i - n(\bar{x})(\bar{y})}{\sigma_x^2 (n-1)}$ | $m = -0.8926$ |
| Ordenada origen $b$ | $b = \bar{y} - m\bar{x}$ | $b = 31.9160$ |
| **Ecuación de la regresión lineal** | $y = -0.8926\,x + 31.9160$ | |

*Figura 14: Cálculo de los diferentes parámetros*

La pendiente se encuentra con esta fórmula que está aquí. La sumatoria de la multiplicación de "x" por "y", que es la que teníamos aquí arriba: 4800, menos n veces la "x" promedio, "y" promedio entre la varianza de la de la "x" por n menos uno porque es una muestra, y nos da esta pendiente. Y la ordenada nos sale con esta fórmula, y nos da esta ordenada.

Ahora, nuestra ecuación es de la forma y=mx+b, entonces "y" es igual a "m" veces "x" más "b", y esta es mi ecuación que nos va a permitir hacer la regresión lineal.

Ahora, ¿Qué hago con esa ecuación? A esa ecuación le podemos asignar múltiples puntos en "x". Y aquí está, se genera una tablita. Este es el valor de "x" que va a entrar en esta ecuación y produce esta "y". Ahora este valor produce este, y así sucesivamente. Entonces, si yo tengo todos estos puntos y los grafico, claramente se ve que es una línea recta porque era la ecuación de una recta.

Pero ¿cómo se verían en conjunto una con respecto al otro? Bueno, más abajo incluimos esa imagen. En esta imagen se ve que está todo el conjunto de puntos del modelo de regresión lineal, que es este verde, y todo el conjunto de puntos de la gráfica de dispersión que eran los datos que

teníamos. Como pueden ver algunos de los puntos coinciden con la recta, algunos están separados, como por ejemplo este de aquí que está muy distante o este de acá.

Hay algunos que no están sobre la recta pero pasan muy cercanos como estos dos o este. Entonces, ¿para qué nos sirve la regresión? La regresión lo que nos hace es darnos un modelo que nos permita aproximarnos a los datos generando ecuaciones, en este caso el de una línea recta, que nos permitirá en un futuro predecir qué valor de la variable dependiente "y" obtendré al introducir un valor de la variable independiente "x" que antes no se había presentado.

Pero bueno, vamos a continuar más adelante haciendo ya la parte práctica en Python. Vamos a ver cuál es la alternativa para evitar todo este trabajo y van a ver que es muy fácil y cómo se debe usar aún si no tienen conocimientos...aún si no tienen conocimientos previos del área de matemáticas o de geometría analítica.

## Práctica del modelo de Regresión

Una vez que ya vimos la parte teórica de la regresión lineal ahora vamos a ver cómo se va a hacer en nuestro software Python. Para ello vamos a continuar en el mismo cuaderno con lo que estuvieron viendo la teoría. Estaba aquí abajo y vamos a empezar la parte práctica. Ahora, qué conjunto de datos vamos a utilizar. Vamos a utilizar el mismo que habían estado usando de la predicción de casas o de costos de casas.

Para ello vamos a utilizar el load boston otra vez. Con load boston vamos a cargarlo a la... como función y lo guardamos en una variable boston_dataset. Ahora, viene un formato de difícil de trabajar, por lo tanto hay que traerlo a Pandas.

Vamos a decir: Pandas, data frame, le vamos a pasar esa variable, boston_dataset, y nada más le vamos a pasar los datos, pero como títulos de columnas vamos a pasarle de esa misma variable el feature_names, que son los títulos de las columnas.

```
from sklearn.datasets import load_boston

# Cargamos un conjunto de datos

boston_dataset = load_boston()

# Se carga con pandas para tratamiento posterior

import pandas as pd

df =
pd.DataFrame(boston_dataset.data,columns=boston_dataset.feature_na
mes)

# Agregamos la variable de respuesta

df['MEDV'] = boston_dataset.target[df.index]

df.head()
```

Ahora sí, ya tenemos nuestro conjunto de datos listo. Nada más nos falta agregarle la variable de respuesta, recordemos que viene en otra... en otra variable. Vamos a decirle "boston_dataset" y le vamos a decir "target", y le vamos a dar el index. Ok. Y eso nos genera nuestra variable de

respuesta, y si lo generamos ya tenemos nuestro conjunto de datos listo. Recordemos que head nada más nos imprime los primeros 5 registros. En realidad es de un tamaño mayor.

Lo podemos hacer con un df.shape, y nos da 506 registros. Ok. Recordamos que en un video anterior utilizábamos el df.corr el cual nos daba la función de correlación o la matriz de correlación.

```
df.corr()["MEDV"].sort_values()
```

De esta forma podemos... podíamos decidir cuáles variables eran convenientes para realizar una regresión y cuáles no. En vez de estar imprimiendo toda la matriz completa el regalo que nos funcionaba, o lo que queríamos, era la relación que tienen las variables independientes con la dependiente.

Entonces nosotros podemos pasarle o restringirle de esa manera, nada más dame las correlaciones con la variable de respuesta que es el costo promedio de las casas, y vamos a ordenarlas de menor a mayor.

Entonces, lo imprimimos y nos da esta lista. Ahora, ¿cuáles son las variables que nos interesan? Bueno, las que nos interesan son las que son o muy negativas, o sea, las que están hasta acá arriba, o muy positivas, las que están acá abajo.

Este de aquí que nos da un 1 perfecto no nos sirve porque posee la misma variable de respuesta. No podemos predecir el costo de las casas con el costo de los casas. No tiene sentido. Entonces esta la ignoramos. Podemos tomar

estas de aquí y estas de acá. Ok. Vamos a utilizar LSTAT que es la que tiene una correlación de mayor magnitud con 0.73, aunque sea negativo nos interesa la magnitud, no el signo.

Entonces, vamos a tratar de hacer una regresión. Le vamos a decir "de df, LSTAT, que es la variable que nos interesa, dame los valores y dales otra forma con el número de líneas indefinido y de una columna, y lo mismo con la variable de respuesta. Lo vamos a guardar como las variables "x" y "y" respectivamente. "X" se acostumbre a ponerla mayúscula y "y" en minúscula.

```
# Preparamos datos
X = df["LSTAT"].values.reshape(-1, 1)
y = df["MEDV"].values.reshape(-1, 1)
```

Así es, así se ve en muchos lados. Entonces hay que seguir esa conveniencia. Lo ejecutamos y tenemos nuestros conjuntos de datos listos..

Ahora, se mencionó en una sección anterior que hay que hacer un split de datos. Vamos a utilizar la misma función que utilizamos antes: train_test_split, vamos a hacer un test de tamaño de 33% y vamos a poner una semilla aleatoria que sea el número 100 por poner un número.

```
#Creamos un split de datos
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)
```

Y lo generamos, y ahora sí ya tenemos nuestros conjuntos de datos organizados. Vamos a imprimir uno nada más para ver si es cierto: X_train.head( ).

No es cierto... Esto ya no es un arreglo de Pandas. Entonces, lo que podemos hacer es decirle: dame los cinco primeros elementos. Ahora sí, ya lo tenemos 6.59, 5.57. Ok. Parece que está bien.

Ahora a hacer la predicción. Para ello vamos a utilizar la función LinealRegression, o que es regresión lineal, y que viene de la paquetería scikit-learn. Simplemente la vamos a llamar con unos paréntesis para decir que se ejecute y le vamos a decir: entrena con el conjunto de entrenamiento en su "x" y en su "y".

```
from sklearn.linear_model import LinearRegression

#Creando el modelo y entrenando

reg = LinearRegression().fit(X_train,y_train)

#Prediciendo valores de entrenamiento

y_train_hat = reg.predict(X_train)

#Prediciendo valores de validación

y_test_hat = reg.predict(X_test)
```

Eso nos va a dar un modelo que lo vamos a nombrar "reg", de regresión. Una vez que tenemos ese modelo construido vamos a realizar las predicciones, vamos a predecir con X_train y vamos a predecir con X_test. Lo vamos a guardar en estas variables que tienen el mismo nombre de las "y" pero tenemos el agregado "_hat" que es la identificación para decir que son predicciones y que no son datos reales, son predicciones de nuestro modelo.

Vamos a ejecutarlo. Y en este momento ya tenemos el modelo y las predicciones del modelo, pero pues no nos muestra nada. Vamos a graficarlas, para ello vamos a utilizar el matplotlib, que es la librería que nos va ayudar a graficar, y NumPy que la librería para hacer arreglos y funciones de ese tipo, operaciones con arreglos. También hay que decirle, recordemos, a Jupyter, que queremos que las gráficas las imprima en el mismo cuaderno y no en alguna parte externa.

```
import matplotlib.pyplot as plt

import numpy as np

# Le decimos a jupyter que grafique en  el cuaderno

%matplotlib inline

# Creamos un scatter plot con los datos de entrenamiento

plt.scatter(X_train, y_train)

# Creamos un scatter plot con los datos de validación

plt.scatter(X_test, y_test)

# En X_plot guardamos valores distribuidos entre 0 y 40

X_plot = np.linspace(0,40).reshape(-1, 1)

# Con el modelo predecimos X_plot

y_plot = reg.predict(X_plot)

# Graficamos el modelo
```

```
plt.plot(X_plot, y_plot,"r--")
```

En primera instancia creamos nuestro primer scatter que va a imprimir solamente los datos de entrenamiento, y como no le asignamos un color, el color de default seguramente va a ser azul. Lo volvemos a hacer con los datos de prueba que va a poner puntos en naranja, y le vamos a decir a NumPy que nos dé un arreglo o una lista con valores distribuidos entre 0 y 40. ¿Por qué esos valores? Porque pues yo ya sé de antemano que son los rangos que maneja LSTAT.

Vamos a decirle que esos valores nos los de en una forma donde el número de filas no nos interesa pero queremos que sea una columna. Y esa variable, que le vamos a llamar X_plot por conveniencia, se lo vamos a pasar a nuestro modelo para que haga una predicción con eso, y nos va a dar las variables de respuesta, y esos dos que acabamos de crear vamos a graficarlos. Entonces lo obtenemos y listo, tenemos nuestra gráfica preparada.

Ahora, vean que son muchos puntos más que los ejemplos que habíamos estado usando antes porque ahora estamos usando el conjunto de datos completos, ya es un programa un poquito más difícil, y por lo tanto, si calculamos "r^2" como lo hicimos en videos pasados seguramente el error es menos relevante que la vez anterior.

```
from sklearn.metrics import r2_score

print("Entrenamiento", r2_score(y_train, y_train_hat))

print("Prueba", r2_score(y_test, y_test_hat))
```

Nos da un 0.54 en entrenamiento y 0.53 en prueba. Son valores muy bajos para decir que es un modelo confiable.

Entonces, ¿cómo podemos mejorar esto? Bueno, si recuerdan solamente utilizamos la variable LSTAT para hacer la predicción, pero tenemos otras muchas columnas y puede que tengan información muy valiosa. Recordemos que LSTAT es la única que tiene un valor de correlación muy grande pero también tenemos RM que es el número de cuartos o, por ejemplo, tenemos PTRATIO que también tiene un buen valor de correlación.

Entonces, hay otra información en el conjunto de datos que no estamos aprovechando. De ahí viene la idea de extender la regresión lineal a una regresión múltiple.

## Regresión múltiple

Una manera natural de mejorar estos resultados es utilizar más variables independientes para poder así mejorar la predicción. Esa es la idea detrás de la regresión múltiple, con ella vamos a extender las capacidades de nuestra revisión lineal y vamos a poderlo hacer de una manera muy sencilla y muy similar a como lo hacemos en la regresión lineal.

De hecho, todas nuestras clases son las mismas, lo único que va a cambiar es cómo estamos alimentando nuestro conjunto de datos.

Veámoslo. Vamos a continuar en el mismo cuaderno del video pasado, que es regresión lineal, de hecho; está inmediatamente después. La regresión múltiple es una extensión natural de lo que es la regresión lineal, es muy sencillo de utilizar en Python, solamente tenemos que cambiar la variable "X" que es nuestro conjunto de datos de variables independientes.

Entonces, cambiemoslo un poco, por ejemplo; aquí tenemos el número de cuartos y tenemos la edad.

```
# Preparación de datos iniciales

X = df[["RM","AGE"]]

y = df["MEDV"].values.reshape(-1, 1)


# Hacemos un split de 33%

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=100)


# Se entrena el modelo

reg = LinearRegression().fit(X_train,y_train)


# Hacemos las  predicciones

y_train_hat = reg.predict(X_train)

y_test_hat = reg.predict(X_test)


# Calculamos el error

print("Entrenamiento", r2_score(y_train, y_train_hat))

print("Prueba", r2_score(y_test, y_test_hat))
```

Vamos a cambiarlo por "LSTAT" y vamos a agregar "edad". Ok. Vamos a hacer un train_test_split igualito como lo hicimos antes, incluso damos la misma semilla, hacemos la regresión lineal y la entrenamos, después hacemos las predicciones y calculamos el error. Todo es lo mismo a la sección pasada, la única diferencia es que cambiamos esta parte de aquí: la "x". Si se dan cuenta ya no estamos haciendo un reshape. ¿Por qué? Porque ya tenemos un número de filas y columnas definidos, ya no hay que cambiarlo. Entonces, lo corremos y nos da un error de 0.55 en r^2". Si comparamos con el ejemplo anterior pues el error mejoró pero no tanto. Es muy pequeño en la diferencia. Entonces, qué tal si ya no utilizamos LSTAT, qué tal si utilizamos RM y edad, lo probamos y en realidad empeoró el resultado. Ahora si se fijan en prueba es 0.49 cuando era 0.53.

Entonces, ¿por qué bajó? Bueno, lo que pasa es que "edad" en realidad no nos está ayudando tanto. Si utilizamos RM y LSTAT fíjense como el error empieza a mejorar, pasó de los 50 a los 60. Entonces, ¿qué pasa si empezamos a utilizar, en vez de dos variables nada más... utilizamos todo el conjunto de datos?

Vamos a utilizar todo menos la variable de respuesta, obviamente, verdad. Entonces, vamos a decir df, le vamos a eliminar la columna MEDV, que es la variable de respuesta, y para decirle que son columnas le tenemos que decir que es en el eje 1, que corresponde a las columnas. Entonces, si volvemos a ejecutar, nos da un error mucho mejor. Nos da 0.75 y 0.69. ¿Por qué? porque ya está

usando todas las columnas, porque está usando todas las columnas, entonces el error tiende a mejorar. Ok.

Ahora, pero ¿qué pasa si quitamos algunas de las variables que no tienen tan buena correlación? Tomemos alguna. Por ejemplo, tenemos CHAS, este de aquí. Vamos a quitársela. Le vamos a decir: en vez de que sea nada mas MEDV le vamos también a eliminar esta variable, lo hacemos y en realidad no varió tanto el error. No fue muy significativo el cambio que realizó. Entonces, una parte importante en la selección de variables, en la regresión múltiple, es estar haciendo ese equilibrio entre qué variables sí importan y qué variables no importan de nuestro modelo. Esto nos puede permitir ya a niveles mucho más... mayores de datos hacer predicciones mucho más eficientes utilizando menos datos.

Esa es una tarea indispensable a la hora de estar realizando este tipo de modelos de regresión múltiple. Esto vendrá más adelante, haremos unas secciones especiales para decir cómo podemos seleccionar variables, ya sea a través de de selección escalonada o escalonada inversa, o algún otro tipo de selección que les vayamos a mostrar. Pero eso ya lo veremos más adelante.

Entonces, qué podemos decir de la regresión lineal. Bueno, la regresión lineal es el modelo introductorio a este tipo de problemas, es el modelo más simple, entrecomillado, a utilizar. En realidad todos los modelos son fáciles de utilizar pero pues es el que es más fácil de calcular, más fácil de entender.

En la regresión múltiple... es una extensión natural de la regresión lineal donde en vez de utilizar una sola variable utilizamos múltiples variables y la selección de variables en este tipo de problemas es algo indispensable que estaremos atajando más adelante.

## Conceptos básicos de la Regresión Polinomial

En esta sección vamos a empezar a analizar qué es la regresión polinomial. Para eso preparamos una libreta que, igual que en el caso de regresión lineal, cuenta con la mitad teórica y la mitad práctica. Juntos vamos a empezar a analizar qué es la regresión polinomial desde un punto de vista teórico y te anticipo: el método que vamos a presentar es el método más sencillo para realizarlo a mano.

No necesariamente es el que Python va a trabajar pero esto nos va a dar un contexto general de qué es lo que se está haciendo. Al fin y al cabo lo que se está haciendo es un método numérico que aproxima el comportamiento de ciertos puntos a el comportamiento de un polinomio, pero bueno... no me entro en más detalles hasta que entremos a la libreta.

Para esto vamos a ir a la carpeta a abrir la que dice la "regresión ponomial". Nuestra libreta, igual que en el caso anterior ya les comentaba, tiene una parte teórica con un ejemplo y una parte de práctica en Python. Pero bueno, vamos a empezar en orden por la parte teórica. ¿Qué es una regresión polinomial? Bueno, ya vimos qué es una regresión lineal: es aproximación a un modelo de una ecuación lineal de la forma y=mx+b, que se asemeja mucho al comportamiento de nuestros datos. En nuestro caso actual lo que vamos a tener es un polinomio que se va a acercar mucho al comportamiento en los nuevos datos.

¿Cuáles son las ventajas que tiene un polinomio contra una línea recta? La línea recta, como su nombre lo dice, siempre es una recta, entonces siempre tiene la misma inclinación y entonces

siempre tiene el mismo comportamiento. Si vas de bajada siempre va de bajada, si va de subida siempre va de subida.

Un polinomio lo que le agrega a la hora de estar metiendo términos al cuadrado, al cubo, a la cuarta, le da ciertas curvaturas o cierta flexibilidad. Entonces yo ya no voy a estar atado a una línea recta, ya puedo hacer líneas que bajen y después suban, o líneas que siempre solamente tengan un cambio o al revés. Bueno, para los que tienen conocimientos de matemáticas un polinomio no debe ser algo muy nuevo, para los que no, no se preocupen. Este curso no asume que son expertos en mate. De hecho, no es algo que estemos persiguiendo. Estamos persiguiendo formar científicos de datos que maneje Python. Simplemente esta sección, al menos la parte teórica, es para darles ese conocimiento background para la gente que les interesa, y para la que no simplemente un conocimiento previo que se puedan imaginar qué hace Python internamente.

Entonces, vamos a nuestro ejemplo. Nuestro ejemplo... ahora contamos con siete puntos. Estos siete puntos tienen un par de valores en "x" y un par de valores en "y", que a la hora de graficar estas coordenadas nos da una gráfica como la que tenemos aquí. Esa gráfica de dispersión tiene siete puntos naranjas, ahí está. Para las personas que ya están familiarizadas con el área geometría, eso se asemeja mucho el comportamiento de una parábola. De hecho casi hacen una parábola, pero bueno, para los que no, se las presentaré ahorita que salga. Entonces, de antemano sé que la forma a la que se asemejan mucho una parábola... que es un polinomio de grado 2.

Es decir, la potencia más grande que van a encontrar es una X^2. El polinomio del cúbico sería X^3, cuarta, X^4, y así sucesivamente. Bueno, ahorita vamos a centrarnos en cómo hacer un polinomio cuadrado que se asemeje mucho a ese término.

Para eso vamos a ocupar la técnica de ajuste que se conoce como ajuste de mínimos cuadrados que empieza con esta ecuación, y esta ecuación que se ve aquí es la sumatoria de elementos que de una diferencia que está al cuadrado.

Entonces, desarrollarla, y desarrollar cada uno de esos elementos, sería un poco complejo porque esta ecuación se tiene que analizar desde el punto de vista de las derivadas parciales.

Tenemos tres variables por ahí desconocidas a0, a1 y a2, que éstas van a ser los coeficientes de la ecuación del polinomio. El polinomio en nuestro caso va a ser algo así como a2, x^3, más a1, x lineal, más a0 igual a la función de x. Pero tenemos tres variables, entonces nuestra ecuación puede variar dependiendo cada una de esas. Para eso, ocupando el conocimiento de ecuaciones diferenciales o de derivadas parciales, podemos encontrar estas tres ecuaciones, yo sé que eso es muy engorroso, sé que eso se aleja completamente del alcance de este curso por motivo por el cual ya les traje las ecuaciones directamente. Entonces, las tres ecuaciones que yo tengo aquí son las que vamos a estar trabajando.

$$s_r = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

Utilizando el concepto de derivadas parciales en conjunto con algebra es posible encontrar las constantes que acompañaran a las tres variables $a_0$, $a_1$ y $a_2$ en el sistema de ecuaciones algebraicas tres por tres. Pero no te preocupes nuestro curso no es un curso de matemáticas, para este ejemplo hemos simplificado las ecuaciones de una manera sencilla y comprensiva, tal como se muestra en el recuadro amarillo a continuación.

$$S_r = \sum_{i=1}^{n} (y_1 - a_0 - a_1 x_i - a_2 x_i^2)^2$$

$$\frac{\partial S_r}{\partial a_0} = -2 \sum_{i=1}^{n} (y_1 - a_0 - a_1 x_i - a_2 x_i^2) \rightarrow (n) a_0 + \left( \sum_{i=1}^{n} x_i \right) a_1 + \left( \sum_{i=1}^{n} x_i^2 \right) a_2 = \sum_{i=1}^{n} y_i$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum_{i=1}^{n} x_i (y_1 - a_0 - a_1 x_i - a_2 x_i^2) \rightarrow \left( \sum_{i=1}^{n} x_i \right) a_0 + \left( \sum_{i=1}^{n} x_i^2 \right) a_1 + \left( \sum_{i=1}^{n} x_i^3 \right) a_2 = \sum_{i=1}^{n} x_i y_i$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum_{i=1}^{n} x_i^2 (y_1 - a_0 - a_1 x_i - a_2 x_i^2) \rightarrow \left( \sum_{i=1}^{n} x_i^2 \right) a_0 + \left( \sum_{i=1}^{n} x_i^3 \right) a_1 + \left( \sum_{i=1}^{n} x_i^4 \right) a_2 = \sum_{i=1}^{n} x_i^2 y_i$$

*Figura 15: Sistema de ecuaciones*

Generamos un sistema de ecuaciones 3 x 3. Entonces, ya pueden ver la magnitud de trabajo que es esto con respecto a la regresión lineal. Hacerlo a mano es engorroso, es tardado, es complejo, pero les tengo una buena noticia: Python no les va a exigir que sepan nada de estos temas, y lo va a hacer por ustedes. Pero bueno, vamos a continuar con el ejemplo. Si son observadores aquí dice: "n" que multiplica a0, más la sumatoria de todos los valores de "x", más la sumatoria de todos los valores de "x^2".

Estamos hablando de muchas sumatorias, entonces, para no batallar las plasmamos en una cómoda tabla. En la tabla tengo una columna con los valores de "x", una tabla con los valores de "y", la tabla con los valores de "x" al cuadrado, "x" cúbica, "x" cuarta, "xy" y "x" cuadrada "y".

| Elemento | $x$ | $y$ | $x^2$ | $x^3$ | $x^4$ | $xy$ | $x^2y$ |
|---|---|---|---|---|---|---|---|
| 1 | -3 | 7.5 | 9 | -27 | 81 | -22.5 | 67.5 |
| 2 | -2 | 3 | 4 | -8 | 16 | -6 | 12 |
| 3 | -1 | 0.5 | 1 | -1 | 1 | -0.5 | 0.5 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 3 | 1 | 1 | 1 | 3 | 3 |
| 6 | 2 | 6 | 4 | 8 | 16 | 12 | 24 |
| 7 | 3 | 14 | 9 | 27 | 81 | 42 | 126 |
| $n = 7$ | 0 | 35 | 28 | 0 | 196 | 28 | 233 |
| | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| | $\sum_{i=1}^{n} x_i$ | $\sum_{i=1}^{n} y_i$ | $\sum_{i=1}^{n} x_i^2$ | $\sum_{i=1}^{n} x_i^3$ | $\sum_{i=1}^{n} x_i^4$ | $\sum_{i=1}^{n} x_i y_i$ | $\sum_{i=1}^{n} x_i^2 y_i$ |

Al sustituir los valores obtenidos en cada una de las sumatorias en las ecuaciones del recuadro amarillo, se obtiene un sistema de ecuaciones algebraicas que se pueden resolver por múltiples métodos con lo que se obtienen los valores de cada una de las tres variables, como se muestra en la siguiente tabla.

$$\begin{aligned} 7a_0 + 0a_1 + 28a_2 &= 35 \\ 0a_0 + 28a_1 + 0a_2 &= 28 \\ 28a_0 + 0a_1 + 196a_2 &= 233 \end{aligned} \quad \rightarrow \quad \begin{aligned} a_0 &= 0.571 \\ a_1 &= 1 \\ a_2 &= 1.107 \end{aligned}$$

$$f(x) = 1.107x^2 + 1x + 0.571$$

*Figura 16: Solución de las ecuaciones*

¿De dónde salieron esos elementos? Son los que van pidiendo aquí: "x", "x" cuadrada, "x" cúbica, "x" cuarta, aquí está, "y", que estaba aquí, "x" por "y" que está acá, "x" cuadrada "y" que está acá, y al final nos dicen que lo sumamos. Aquí están los valores de los... todos los que les había dicho. Al sustituir estos datos en nuestras ecuaciones "n" es 7, entonces va a aparecer 7 a0, más la sumatoria de a1, que es 0, más esta sumatoria que multiplica a2, que es 28, pero bueno, saldría un sistema de ecuaciones parecido a este, y este sistema de ecuaciones se puede resolver de muchas formas diferentes, desde métodos sencillos como el gráfico, desde métodos normales y comunes como el de sustitución, hasta métodos un poco más sofisticados como el de Gauss-Jordan o el de determinantes, no importa, el cómo lo hayan resuelto no es de la importancia de este curso, pero las soluciones de este sistema de ecuaciones serían estas tres soluciones que la hora de plasmar los en el polinomio sería a0 por "x" cuadrada, más a1 por "x", más a0, y aquí están.

Esta ecuación de este polinomio de segundo grado a la hora de asignarle valores de "x" produce valores de "y", y aquí tengo unos cuantos, 20 puntos, para darnos una idea de cómo se va a comportar, y aquí está la parábola. Ahora, si son observadores y recuerdan la forma de la otra o van arriba, los datos se parecen mucho.

¿Qué tanto se parecen? Más abajo tenemos una foto en la que ya están los valores en conjunto, aquí tenemos en verde el modelo que encontramos después de mucho trabajo algebraico, y tenemos en naranja los puntos de la dispersión. Entonces, hay muchos puntos que se parecen muchísimo y hay puntos que están un poco alejados como este de aquí.

Bueno, el objetivo de esta regresión no es hacer un modelo que sea exactamente idéntico a todo, sino, encontrar un modelo que se aproxime de forma buena, de forma consistente, a la mayoría de los puntos. Más adelante vamos a trabajar con, no sólo regresiones cuadráticas, sino cúbicas,

cuartas, de más potencias, pero ya hemos hablado en videos anteriores de que eso también puede provocar el problema que se conoce como sobreentrenamiento.

Entonces, a medida que tu experiencia vaya creciendo o que empieces a meter métodos como cross validation y cosas de ese estilo, tú podrás irte haciendo de una idea, generalmente producto de la experiencia, de qué grado del polinomio requieres para asemejarte lo más posible a los datos sin incurrir en ese tipo de problemas.

Por mi parte, la parte teórica, es todo. Espero que continúes con esta misma libreta en la parte práctica.

## Practicando la Regresión Polinomial

Vamos a continuar con la parte práctica de la regresión polinomial una vez que ya vimos un poquito la teoría. En realidad la regresión polinomial en Python difiere un poco de la teoría que vimos anteriormente. Es una manera más práctica de hacer el cálculo de una regresión polinomial, y lo hacemos de una manera muy sencilla.

En realidad lo que hace Python es un preprocesamiento de los datos. A nuestros datos originales les aplica una transformación la cual los convierte de un valor numérico a un polinomio. De esta manera tomamos los datos originales, que son las variables independientes, y las transformamos a un polinomio. De esta forma, ya que la tenemos en forma de polinomio, Python puede aplicar su fórmula, o su librería de regresión lineal, y ahora sí puede predecir de manera efectiva a través de un polinomio.

Ahora, vamos a utilizar el mismo conjunto de datos de la vez pasada para no perturbar esa área, solamente concentrarnos en cómo se hacen las predicciones. Vamos a utilizar el load Boston otra vez, vamos a crear nuestro data frame y lo vamos a imprimir. Igual como en la sección anterior. Vamos a obtener las mayores correlaciones que ya sabemos de antemano que es LSTAT, RM y por ahí también está PTRATIO, y vamos a hacer una regresión con una variable. Para ello vamos a utilizar el LSTAT que es el que hemos estado utilizando comúnmente.

Vamos a preparar esa información como "x" y como "y". X mayúscula, recordemos, y vamos a realizar un split de los datos.

Vamos a utilizar la misma configuración: 33% y las semilla de... como 100, la hacemos y ahora sí vamos a utilizar la revisión polinomial junto con la regresión lineal.

Para la regresión polinomial, así como en cross validation utilizábamos un parámetro k, aquí necesitamos un parámetro el cual nos indica el grado de la regresión polinomial. Esto lo definimos como degree, y vamos a utilizarlo como PolynomialFeatures, que es la clase que nos va a transformar de valor numérico a polinomio. Para ello hay que utilizarlo en alguna variable. Vamos a llamarle poly por el momento, y poly lo usamos para transformar los datos originales, como dijimos, que los datos originales es x_train y x_test, y lo vamos a transformar en su forma polinomial.

Esto lo vamos a ejecutar y nos va a generar la transformación. Luego con una regresión lineal vamos a entrenar pero sobre los datos transformados, no sobre los originales. Vean, aquí dice X_train.poly que esté de aquí, y Y_train que son... esa sí es la variable original, la variable de respuesta, esa no la transformamos en polinomio, y eso nos genera un modelo "reg".

Ahora, este reg, aunque es muy similar al que usamos antes, hay que tener la consideración que siempre que lo vayamos a utilizar los datos que le vayamos a incrustar o a alimentar hay que transformarlos primero a polinomio. ¿Por qué? porque así fue entrenado. Una vez que lo entrenamos sólo va a recibir ese tipo de información.

Entonces, hay que ser consistentes. Una vez que ya lo tenemos entrenado podemos empezar a hacer predicciones. Recordemos, como les acabo decir, que para darle información hay que transformar en polinomio. Entonces, lo vamos a predecir con X_train pero en su forma polinomial, y X_test en su forma polinomial, y esto lo guardamos como y_train.hat, que es... recordemos, "y_train.hat" que es el que nos dice que es una predicción. Entonces, es la predicción de la "y" en entrenamiento y la predicción de "y" en prueba. Ok.

Lo hacemos y nos marcó un error. Me faltó ejecutar esta celda de acá arriba. La ejecuto y, otra vez... listo. Ahora, ya lo tenemos, pero pues hay que verlo de alguna forma. Si nada más lo entrenamos y no hacemos nada con esos datos pues no nos dice nada. Igual que la vez pasada vamos a graficar, vamos a utilizar la librería matplotlib y NumPy, y le vamos a decir a Jupyter que las gráficas las haga en el mismo cuaderno con matplotlib inline, vamos a crear el scatter con la "x" en entrenamiento y la "y" en entrenamiento, y el scatter con x_test o de prueba, y "y" en prueba. Vamos a hacer variables distribuidas entre 0 y 40 ¿para qué? Para que pueda graficar en la gráfica, en el eje "x", y recordemos que para hacer la predicción teníamos que transformar estos datos a polinomio. ¿Cómo? Directamente con la poly que es el que hicimos acá arriba. Aquí está.

Poly. Lo pasamos, estos datos que estamos generando, los transformamos con poly, y ahora sí, tenemos X_plot pero en polinomio. Y con eso ahora sí ya podemos hacer la predicción que nos da "y_poly", y la graficamos. Entonces, nos genera esta gráfica de aquí.

Si se dan cuenta la línea ya no se ve como línea, en realidad es un polinomio de grado 3. Entonces, nos permite producir esta curva de aquí. ¿Qué pasaría si lo dejamos en grado 2? Lo vamos a ejecutar, lo ejecutamos y aquí ya se ve. Va descendiendo, llega a un punto mínimo y luego empieza a subir. Es una parábola, es cuadrática. Esto nos va a permitir asignarle los grados de flexibilidad a nuestra predicción.

Si le agregamos 4 ya se empieza a colocar un poquito más cerca de los datos. Entonces, en una sección anterior les mencionamos que esta variable de aquí es muy importante a la hora de decir si el modelo va a tener un sobreajuste o un muy mal ajuste. Por el momento vamos a dejarlo en 3, vamos a predecir otra vez y vamos a hacer la métrica de error.

Ahora nos da 0.68 y 0.61. Vamos a repetirlo pero con un polinomio de grado 2. Preparamos los datos iniciales de la misma manera, hacemos un split de 33%, hacemos el procesamiento de los polinomios, que es con poly, hacemos la transformación y tenemos nuestra "x" original pero ahora transformada. Hacemos las predicciones y tenemos métricas de error.

Lo que nos esta diciendo estos errores es de que la parábola o el cuadrático no es tan eficiente como el cúbico que tenemos acá arriba. Entonces, vamos a cambiarlo a cúbico. Vamos a ver, ¿empeora o mejora? Mejoró. Vamos a cambiarlo a cuadrático, en cuadrático, ¿mejora o empeora? Mejoró… y mejoró. Ok.

Lo seguimos incrementando y sigue mejorando. El problema de esto es que tal vez está haciendo ya... empieza a acercarse a un sobreajuste de los datos. Seguramente ya lo está haciendo, y lo que

estamos pasando es que el x_test al estar probando constantemente con el mismo conjunto de entrenamiento estamos asignando aprendizaje de forma indirecta a nuestro modelo.

La manera más adecuada de realizar este tipo de pruebas es a través de un cross validation que ya lo vimos en una ocasión anterior. Entonces, el juego, como reto, con este conjunto de datos que tenemos aquí y con lo que vimos en una sección anterior intenten hacer un cross validation y vean cuál es el error real, y con el cross validation traten de hacer un ajuste de esta variable de aquí. Del grado del polinomio. Esa es una práctica que les va a ayudar a aprender dos cosas fundamentalmente, que es cross validation y empezar a hacer ajustes de parámetros. Esto es una actividad que es muy importante para un científico de datos porque no puede nada más construir un modelo y ponerlo en producción.

Hay que estarlo probando, y la manera adecuada de hacer esto es a través de un cross validation. Con esto podemos concluir la regresión pronominal, es una manera muy sencilla de crear modelos ya eficientes que sean no lineales. En la regresión lineal tú tienes que suponer que los datos se asemejan a una línea. En la regresión polinomial no, pueden tener diferentes formas y con la polinomial te puedes ajustar a ellos.

Entonces, es un modelo muy muy poderoso en cuanto es la creación de modelos. A parte de la regresión lineal existen modelos como SVR o árboles de regresión que veremos en secciones más adelante.

## La teoría fundamental del SVR

En esta sección vamos a empezar a ver otro método de regresión. Hasta ahora ya vimos la regresión lineal, la regresión múltiple, la regresión polinomial.

Ahora toca que veamos las SVR, para eso vamos a abrir nuestra carpeta con las libretas y buscamos la que dice SVR. Igual que los videos anteriores, este video está separado en dos partes; vamos a primero explorar la parte teórica y después tenemos una parte práctica para que pongamos en movimiento lo que aprendimos.

Entonces, vamos a nuestra carpeta, aquí está la SVR, y vamos a abrirla. Entonces, SVR; ¿de dónde viene el nombre? El nombre SVR es su nombre en inglés: Support Vector Regression, Regresión de Soportes Vectoriales. Sé que es un nombre un poco, poco intuitivo, que dices ¿qué es eso de vectores de soporte? No se preocupen, más adelante lo vamos a ir viendo en esta libreta juntos, pero bueno.

Es un algoritmo de regresión basado en máquinas de soporte vectorial, que comúnmente son utilizadas para clasificar elementos, en este caso no vamos a clasificar, vamos a ocuparlo para hacer una regresión, pero bueno; vamos a ver cómo funciona una SVR. Aquí les quiero aclarar algo, la SVR no solamente funciona para datos linealizados, pero, por conveniencia de la explicación rápida; ocupamos un conjunto de datos linealizados porque es fácil de que ustedes lo vean y nosotros lo podamos presentar de una forma amigable, pero no se dejen influenciar por esto, no crean que es la única aplicación, pero bueno; entonces, tenemos este conjunto de datos.

Este conjunto de datos tiene el comportamiento similar a como sería una línea recta, podríamos decir que hay una línea recta que pasa por el medio de ellos que los atraviesan.

Bueno, el primer paso para construir unas SVR es generar un hyper-plano. Un hyper-plano en el contexto de los datos linealizados, sería esta línea que atraviesa la mayoría de los datos, en datos que no están linealizados, sería una figura que atraviesa también todos los datos de una manera sencilla. Pero bueno, entonces en nuestro hyper-plano sería esta línea y de ahí continúa; al ser una línea debe de tener una ecuación, una ecuación de la forma: y = mx + b, algo de geometría vamos, pero bueno; paralela a esta línea que sería, por así decirlo; el cero, el punto medio, va a haber dos líneas que vamos a encontrar que se van a llamar vectores de soporte.

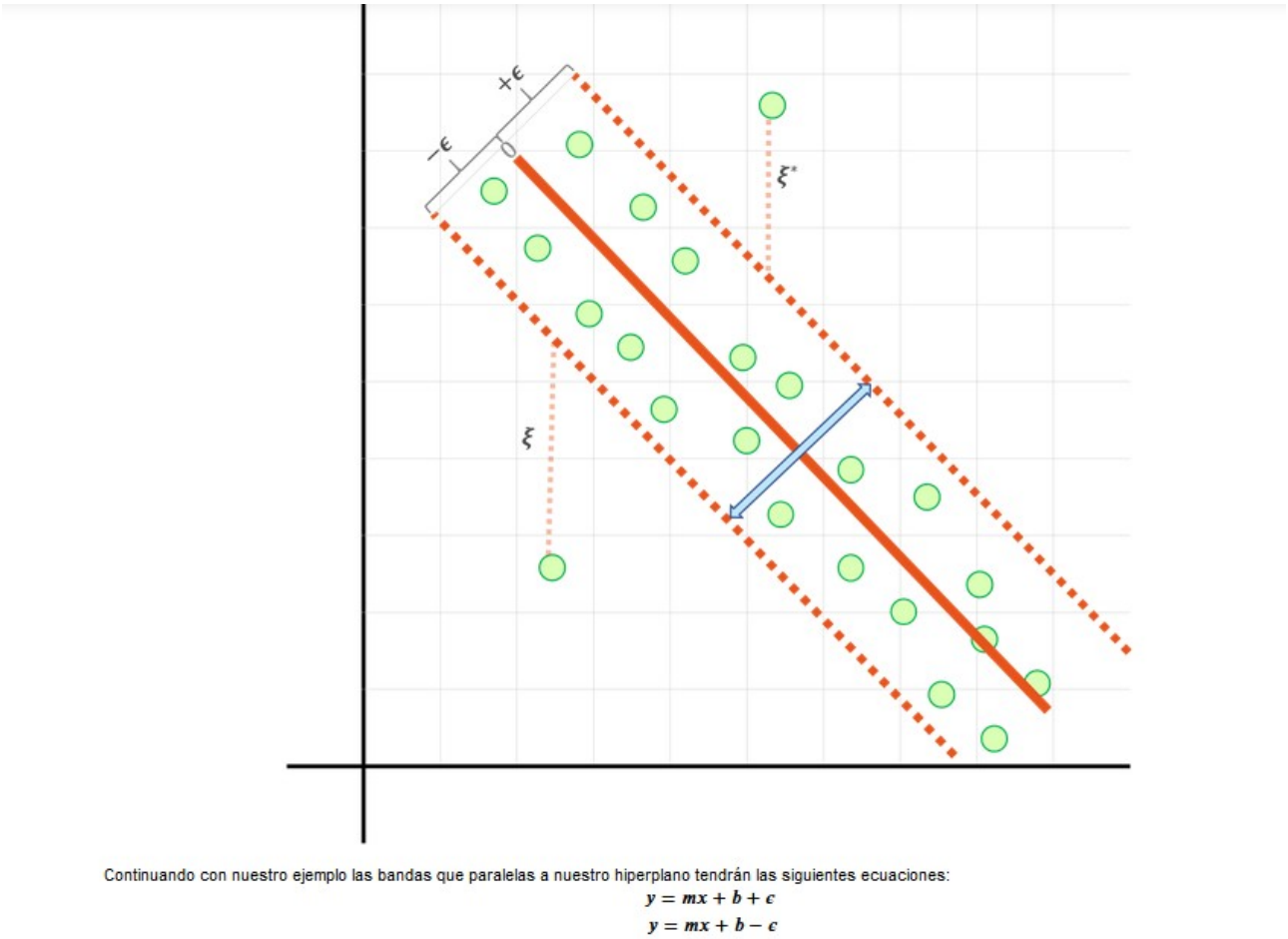


*Figura 17: Vectores de soporte*

Nuestro objetivo es maximizar que la mayoría de los puntos queden dentro de estos vectores de soporte. Entonces esta distancia de aquí se trata de que sea la más grande, pero, minimizar también los errores que queden fuera. Entonces, las ecuaciones de los dos vectores de soporte se vuelven a hacer rectas, entonces simplemente es con este parámetro adicional, y ahora sí; ¿cuál es la ecuación que tenemos que respetar?

Es esta que está en pantalla: minimizar un medio de la magnitud al cuadrado del hyper-plano, más, una constante que multiplica la sumatoria de los errores.

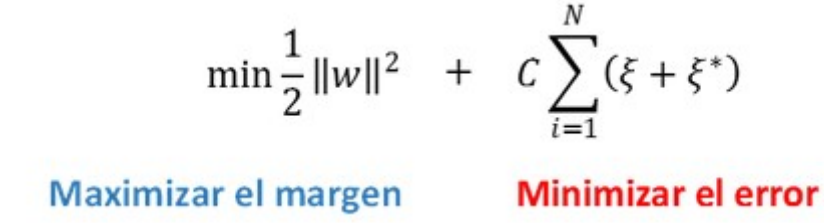


*Figura 18: Ecuación a trabajar*

Entonces en nuestra fórmula se relacionan muchas variables tanto "w" como es la magnitud del hyper-plano, como la "C" que es una constante, que debe ser mayor a cero; que se encarga en controlar el equilibrio entre la regularidad de la función y cuánto toleraremos la desviación fuera de las bandas de soporte, y las variables "XI" esta letra griega se pronuncia así; las cuales controlan el error cometido por la función de regresión al aproximar las bandas.

El cálculo interno que se realiza con las máquinas de soporte vectorial, realizarlo a mano es un poco complejo, o requieren matemáticas que escapan un poco de lo que nosotros proyectamos como un necesario para entender el curso. Pero Python lo puede hacer automáticamente.

En esta libreta en particular, solamente se puso el ejemplo para que vieran cómo se construía, se hagan una idea, pero más adelante ya vamos a empezar a usar una librería especializada donde ya puede hacer las funciones automáticas, ya construye, uno no necesita saber qué tipo de hyper-plano se va a ocupar, o el kernel que va a ocupar, esas palabritas van a tomar sentido a medida que lo estemos practicando, pero bueno.

Por ahora la explicación simplemente es generar un hyper-plano que abarque la distribución o se asemeje mucho a la distribución de los puntos y generar unos soportes que sean paralelos a ese hyper-plano, que sean estos márgenes que van a tratar de abarcar la mayor cantidad de los puntos dentro.

## Práctica de SVR

En esta sección vamos a continuar con la parte práctica de lo que es SVR, y vamos a ver una pequeña demostración de cómo funciona. Esto lo vamos a hacer con el mismo conjunto de datos que usamos la vez pasada para la regresión.

Entonces, vamos a utilizar el cuaderno SVR. Vamos a llegar hasta la parte teórica y de aquí vamos a comenzar, como se los menciono vamos a utilizar el mismo conjunto de datos de la vez anterior, esto ya lo hicimos pero vamos a repetirlo. Vamos a utilizar sk-learn, data sets y vamos a importar load boston.

Este ya viene con el conjunto de datos de precios de casas. Lo vamos a cargar llamándolo con paréntesis, y con Pandas lo vamos a pasar a un data frame porque vienen los datos crudos y lo queremos de una forma presentable. Entonces lo vamos a querer con Pandas, le pasamos el punto data y las columnas o los titulares de arriba pues van a ser los feature names. Ok, lo agregamos como df y de ahí sólo nos falta agregar la variable de respuesta, que es el costo de las casas, verdad, y eso lo hacemos con la variable target. Lo ejecutamos. Tarda un poco porque está cargando los datos, e incluso también está cargando Pandas.

Y ya lo tenemos listo. Ahora con esto vamos a empezar a manejar todo lo que son las correlaciones, a hacer las predicciones, vamos a utilizar la correlación... la correlación de todo contra la variable de respuesta, y lo vamos a ordenar.

Ya tenemos que LSTAT tiene una gran magnitud de correlación con respecto a la variable de respuesta. También PTRATIO tiene 0.50, e incluso también tenemos por acá RM que, si recordamos, es el número de cuartos de la propiedad.

Entonces, a partir de ahí ya podemos obtener qué datos vamos a utilizar, nuestra "x" y nuestra "y" para prepararla, para utilizarla con la SVR. Recordemos que las variables independiente las vamos a guardar como "x" y la variable dependiente como "y". Entonces, vamos a hacer eso, le vamos a decir: df, dame el LSTAT, es la variable que vamos a utilizar en este caso, dame sus valores y cámbialos con esta forma donde el número de rows, o de filas, es indefinido, pero queremos solamente una columna. Eso para "x" y para "y".

Ahora, ya que lo tenemos procesado aquí ya viene algo diferente. Vamos a utilizar un escalamiento de los datos, estos... lo que va a hacer es: si tenemos datos que rondan, por ejemplo, de entre 0 a 100 los va a apachurrar en un rango entre 0 y 1. Esto para facilitar la tarea a nuestra máquina de soporte vectorial. De hecho, aquí abajo viene" tips de uso" es una de las recomendaciones que nos dice... que nos hacen, que hay que normalizar nuestros datos. Este proceso se llama "normalización", y lo que vamos a hacer es... vamos a utilizar el MinMaxScaler. Es una fórmula muy sencilla donde se utiliza el máximo, el mínimo y se divide, y con eso ya obtenemos un rango entre 0 y 1. Pueden chequear más detalles en la documentación de este, pero no vamos a entrar mucho en detalles, simplemente lo vamos a utilizar. Vamos a utilizar MinMaxScaler, tenemos que hacer uno para "x" y otro para "y".

Podríamos usar uno para los dos pero en la experiencia es más fácil utilizar dos por separado sobre todo cuando queremos denormalizar, es decir, pasar otra vez del rango entre 0 1 a un rango más grande.

Entonces, los creamos X_scaler y y_scaler, los inicializamos y luego le decimos a estos mismos que se entrenan o se ajustan, y ahí mismo que se transformen, o más bien, que transformen la información. Entonces, le pasamos como parámetro "x" que es nuestro conjunto de datos original, le decimos ajústate a estos datos y luego transforma "x" al rango que queremos que es 0, 1, y lo vamos a sobreescribir como "x" en la misma variable original. Ok. Entonces ya. Con eso debemos de tener nuestra "x" y nuestra "y" listos para empezar a hacer predicciones.

Pero recordemos que hay que hacer un train, test, split, o sea, una división de los datos, el entrenamiento y prueba. Esto es igual a como lo hemos hecho antes. Entonces, simplemente lo voy a ejecutar. SVR tiene muchos más parámetros que los algoritmos que hemos utilizado con anterioridad. Les recomiendo que se echen un clavado en la documentación que aquí les tenemos preparada y que lean brevemente los tips de uso que ofrece scikit-learn.

Ahora, si hicieran eso notarán que maneja un parámetro llamado kernel, la máquina de soporte de regresión, en este especificamos tres diferentes, vamos a especificar "rbf", el lineal y el polinomial. Rbf es el kernel que más se va a utilizar, o que más se utiliza en SVR, es el predefinido incluso.

Sin embargo también podemos utilizar una regresión, o un kernel lineal, que sería el equivalente a una regresión lineal y un kernel polinomial que sería el equivalente a una regresión polinomial.

Cada uno de estos tiene sus diferentes parámetros que los pueden revisar, repito, en la documentación. Vamos a preparar los tres modelos: un SVR con el rbf, un lineal y un polinomial.

Lo ejecutamos y ya los tenemos listos para empezar a entrenarlos y a partir de ahí empezar a hacer predicciones. Vamos a manejarlos uno por uno, primero con rbf, vamos a decirle: SVR_rbf, que es nuestra primera máquina en soporte, ajústate a X_train y a y_train, y con eso ya podemos empezar a hacer las predicciones. Vamos a utilizar esa misma máquina de soporte, y le vamos a decir: haz una predicción con estos datos, que es X_train, o sea, el conjunto entrenamiento, la variable independiente, el conjunto de entrenamiento y haz lo mismo con la "x" pero de prueba, y eso lo vamos a guardar en sus respectivas variables de "y".

Recordemos que cuando hacemos una predicción tenemos que utilizar este sufijo que es el testado, o hat. Ok. Si lo hacemos hace la predicción pero no imprime nada, simplemente lo ejecutó. No hay nada que mostrarnos en pantalla. Entonces vamos a hacer que nos muestre algo, ¿no? Para visualizar la información.

Entonces vamos hacer la gráfica para empezar a ver qué está haciendo nuestro modelo. Para ello vamos a utilizar matplotlib con "plt" como lo hemos utilizado hasta el momento, vamos a utilizar aparte NumPy, ya lo veremos en un momento para qué lo vamos a utilizar NumPy, y le vamos a decir a Jupyter que las gráficas las cree en el mismo cuaderno o las muestre en "line", o sea, en línea. Ok. Vamos a generar una gráfica de dispersión para el conjunto entrenamiento en "x" y "y", y otro para prueba "x" y "y".

Ahora, aquí es donde vamos a utilizar NumPy. Le vamos a decir: NumPy, por favor genera un conjunto de datos o de valores en el rango de 0 a 1. ¿Cuántos? Bueno pues tiene un valor establecido que es el valor de 50, nos va a generar 50 puntos entre 0 y 1. Esto para poder graficar nuestro modelo, pero los queremos con una forma muy específica.

Los queremos que estén distribuidos en una columna. ¿Por qué? Porque así es la forma que tiene nuestro conjunto de datos de "x". Entonces tiene que asemejar esa misma estructura. La vamos a guardar como X_plot y vamos a hacer el respectivo en "y". Para ello vamos a utilizar el modelo. Le vamos a decir: modelo, haz una predicción con estos puntos que acabamos de generar, y guárdalos como y_plot. Lo graficamos el modelo a través de esta función plot. Y le vamos a decir: x, y, las que acabamos de generar y, en este en particular, vamos a hacer una línea punteada roja. Ok. Lo ejecutamos y nos muestra la gráfica.

Si queremos ocultar este tipo de líneas lo que podemos hacer es darle punto y coma al final y eso le indica a Jupyter que lo último no lo imprima. Entonces, lo hacemos y ya no nos muestra ese mensaje feo, verdad.

Nada más la gráfica que estaba buscando. Ahora, pero la gráfica puede ser difícil de interpretar solamente si la mostramos. Siempre hay que tratar de hacer algunas métricas de desempeño para ver qué tan bien le fue al algoritmo. Entonces, para ello vamos a utilizar el "r2", que es la métrica que hemos estado utilizando. Vamos a decirle que para entrenamiento calcule el r2_score entre y_train, que son los datos originales de "y", pero de entrenamiento, y las predicciones de "y" en entrenamiento-

Algo similar se hace pero, en lugar de entrenamiento, para prueba. Entonces ejecutamos los dos. Nos da un 0.65 en entrenamiento y 0.59 de prueba. Eso lo pueden tomar y compararlo con los

algoritmos anteriores y ver que tal vez lo hemos hecho en este caso. Pero dijimos que íbamos a entrenar con tres kernels diferentes, o tres modelos diferentes.

Ya utilizamos lo que es eñ rbf, ahora vamos a pasarnos al lineal a ver cómo se comporta. Ok, vamos a decirle: SVR lineal, entrena con estos valores. Si se dan cuenta son las mismas instrucciones de arriba. Las mismas instrucciones de arriba... lo único que cambia es el modelo que estamos utilizando, ya no es SVR_rbf, sino SVR_line, de línea. Entonces lo podemos ejecutar. Es exactamente los mismos datos, y nos genera esta gráfica de aquí que claramente nos muestra que se graficaron como una línea.

Bueno, y ¿qué pasa si utilizamos el SVR_poly? Bueno, como podrán imaginar lo que nos va a generar es un polinomio que va tomando la forma de los datos, que se asemeja mucho lo que vimos en videos pasados.

Ok. Bueno, entonces vamos a verlo pero de una forma en la que podamos comparar directamente estos tres SVRs. Vamos a manejarlo en una sola gráfica. Para ello vamos a utilizar los mismos tres kernels: rbf, lineal y polinomial. Vamos a hacer una gráfica de dispersión con los datos de "x" y "y". Nada más que aquí le agregamos un alfa de 0.3, ¿para qué? Para que los puntos de entrenamiento y prueba se vean más difuminados y darle más importancia a los modelos. Ok. También utilizamos con X_plot el conjunto de datos que nos va a generar todos los puntos en "x" para poder visualizar correctamente nuestros modelos.

Ahora, como todo lo queremos en una sola gráfica, lo tenemos que hacer todo en una misma celda de procesamiento. Recordemos que la celda es estos cuadros de código, y lo que va a cambiar es cuál es el algoritmo que estamos utilizando o el modelo, ya sea rbf, lineal y polinomial. Lo demás es idéntico. Entonces, frecuentemente cuando llegamos a una tarea que se ve repetitiva generalmente es indicación de que hay que utilizar un ciclo.

Entonces, para ello vamos a utilizar un ciclo "for". Le vamos a decir: para cada nombre, svr y s, para cada uno de estos, haz el siguiente conjunto de líneas, o ejecuta el siguiente conjunto de líneas. Pero qué representan estos: name, svr y s. Bueno, aquí lo tenemos.

Python tiene una función muy especial, que facilita mucho las cosas una vez que aprendes a utilizarla, llamada "zip". Zip, si saben de archivos comprimidos o de carpetas comprimidas, el zip lo que hace es tomar archivos y los mete en uno solo. O sea, diferentes archivos distribuidos en carpetas y los incrusta en uno solo. Algo similar a hace zip, donde toma tres listas diferentes, esta es la primera, esta la segunda y esta la tercera, y la salina y las compactas en una sola, de forma que cuando las recorremos con un for nos da los primeros elementos de cada lista, luego los segundos elementos de cada lista, los terceros elementos, así hasta el enésimo elemento de cada lista.

Entonces, en cada iteración solamente vamos a leer, ahora sí...qué va a hacer dentro del for. Le vamos a decir: a svr, que en el primer caso es este de aquí, svr_rbf, le decimos: entrena con el conjunto de entrenamiento "X" y con el conjunto de entrenamiento "y".

Una vez ya entrenado podemos empezar a hacer predicciones. Le decimos: haz una predicción con el conjunto de entrenamiento, haz una predicción como el conjunto de prueba, lo mismo... lo que habíamos estado usando antes, y haz una predicción con el conjunto de datos "X" que generamos en un rango de cero y uno, que es el de acá fuera, y eso que acabamos de hacer grafícalo. Le vamos a decir: grafica X_plot y y_plot. y_plot es el que acabamos genera aquí. Le vamos a pasar el argumento "s", que si lo vemos es la tercer variable aquí, que nos dice el tipo de línea que vamos a

utilizar. En el primer caso sería una línea roja, una línea verde y una línea azul, y como último parámetro le vamos a pasar el nombre de ese conjunto de datos que en este caso en específico va a ser rbf. En la segunda iteración va a repetir lo mismo pero ahora con lineal... sbr lineal y verde, y el último va a repetir con polinomial. Todo eso lo podemos generar en misma gráfica, y luego por último nada más utilizamos el plt.legend para agregar la leyenda.

Entonces, esto nos va a generar esta gráfica de aquí, podemos agregar el punto y coma para que no muestre el mensaje ese feo. Ahora sí, tenemos a rbf en entrenamiento, que esto es el primer ciclo, es lo que nos imprimió de aquí.

Genera la línea roja, y ahí termina el primer ciclo. El segundo ciclo es la línea que la general correctamente, o la línea verde, y el polinomial termina siendo la línea azul. Entonces, aquí ya se empieza a ver la diferencia entre los diferentes sbr's.

El lineal pues se ve claramente que no se está adaptando correctamente a los datos, sin embargo el polinomial y el rbf se están adaptando mejor. Entonces, podríamos argumentar por qué decidir uno sobre el otro pero ahorita lo vamos a dejar como el caso de uso, no vamos a decidir cuál se va a quedar. Ok.

Una vez hecho eso pues vamos a repetirlo solamente cambiando algunos detalles para hacer una gráfica que se vea mejor. Entonces, vamos a concentrar todo lo que hemos visto en una sola celda para poder concentrarlo y lo puedan utilizar más adelante, verdad. Vamos a hacer algunos cambios de estilo, vamos a utilizar el seaborn para que se vea mejor que es un estilo, así como una plantilla, o más bien como un tema para el teléfono celular, que le cambia el tema y cambia los íconos, se pueden utilizar temas, en este caso vamos a utilizar seaborn, vamos a definir un tamaño que sea... una gráfica que sea cuadrada y vamos a graficarlo.

Lo demás sigue siendo lo mismo solamente lo concentramos todo el cuaderno en esto de aquí. Ok. Entonces, a partir de aquí ya tenemos nuestra gráfica que ya está cuadrada, podemos visualizar los datos de una manera que le damos una mayor importancia a los modelos, ya se puede ver más claramente las diferencias entre rbf y polinomial sobre todo, y esto no va a poder brindar una mejor toma de decisiones para decir cuál es el kernel que vamos a utilizar.

## El Árbol de Regresión y sus conceptos

Ya exploramos diferentes técnicas de regresión, entre ellas pues la regresión lineal, polinomial, la ACSBR*, pero ahora vamos a ver otro tipo de regresión. Este se llama árboles de regresión. Los árboles de regresión son un método de regresión que se basa en los árboles de clasificación.

Entonces, vamos a, primero, abrir nuestra libreta que se llama "árbol de regresión" en donde la hayan guardado. Igual que las libretas anteriores esa libreta se separa en dos partes: una parte teórica y una parte práctica. Pero en esta en particular la parte teórica, por llamarlo de alguna forma, simplemente es una demostración visual.

Bueno, con ejemplos para que los vean de cómo funcionaría el árbol de regresión. No es tanto los cálculos. Los cálculos internamente ya los van a hacer con la ayuda de Python. Entonces no se preocupen si no saben tanto de estadística o de matemáticas o algo, simplemente para que con apoyo visual agarren la idea de cómo funciona y después entiendan en el momento de ya estar con el libreta en la parte práctica cómo funcionó. Entonces, vamos a abrir nuestra libreta.

Los árboles de regresión son métodos predictivos basados en segmentación en donde el conjunto de datos se divide en diversas categorías y los datos que están dentro de cada intervalo se les asigna directamente un valor numérico establecido. A diferencia de los árboles de clasificación, donde al estar en una categoría se les asignaba una etiqueta, aquí le vamos a asignar un valor numérico. Pero bueno, un ejemplo de árbol regresión. Supongamos que tenemos un conjunto de puntos como el que se muestra en la pantalla, en la pantalla tenemos este conjunto de puntos. Evidentemente... digo evidentemente porque fue hecho así el ejemplo, los puntos están en cuatro conjuntos bien establecidos: el primer conjunto está aquí, el segundo, el tercero y el cuarto.

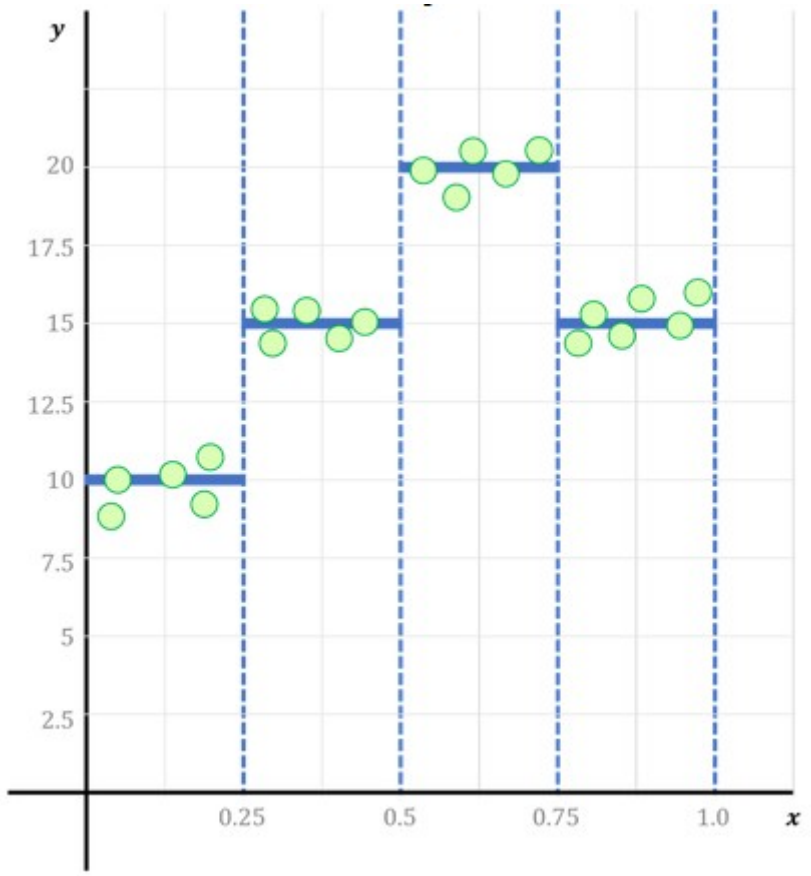


*Figura 19: Puntos clasificados*

No va a ser así normalmente en un ejemplo que ustedes tengan de la vida real. Pero bueno, es la idea lo importante. Entonces, el procedimiento para generar un árbol de regresión empieza partiendo la totalidad del intervalo de los datos, en este caso los datos pueden ser distribuidos desde el 0 hasta el 1, en diferentes subconjuntos. Mientras más subconjuntos tengan el árbol va a crecer.

En este caso en particular nosotros para este ejemplo decidimos usar cuatro intervalos pequeños: el primer intervalo va desde 0 hasta 0.25, el segundo intervalo va del 0.25 a 0.5, el tercero de 0.5 a 0.75 y el cuarto en la parte restante.

Aquí están las ecuaciones de los cuatro intervalos. Ahora, para ejemplificarlo un poco... un poco más agregamos estas líneas azules. Entonces, aquí están los cuatro intervalos bien establecidos. Ahora, todos estos valores que están aquí órbitan o están muy cerca del valor de 10, estos valores que están aquí están muy cerca al valor de 15, estos de 20 y estos repiten el valor de 15. Entonces, quedamos que a los elementos que están en la primera categoría no les íbamos a poner una etiqueta

porque eso sería un árbol de clasificación, sino que les vamos a poner un valor numérico para poder predecir cuando nosotros sepamos el valor de "x" qué "y" van a devolver.

Entonces, nosotros decidimos que todos los puntos que están en el primer intervalo dado que todos están cerca del valor de y=10, a todos les vamos a dar que para el primer intervalo la "y" va a ser 10, para el seguro intervalo la "y" va a ser 15, tercer intervalo la "y" va a ser 20, cuarto intervalo la "y" va a volver a ser 15. Entonces aquí están nuestras ecuaciones.

Pusimos eso... esa información a manera de una línea gruesa azúl para que se vea más clara. Entonces, todos los valores que están en esta región, en ese intervalo, le vamos a asignar inmediatamente el valor de 10 como habíamos dicho. Entonces, ¿por qué se llaman árboles de regresión? Bueno, esa información que les platiqué ahora la podemos, no sólo escribir como ecuaciones o no sólo escribir como texto, sino que la podemos representar en un esquema en forma de un árbol y ese esquema es este de aquí. Este esquema de aquí se abren ramas por cada una de las preguntas que se están haciendo con comparaciones lógicas.

En este caso el valor de "x" es mayor que 0.5, existen dos opciones: que sí o que no. En este caso si no es mayor hacemos otra pregunta, si sí es mayor hacemos otra pregunta. Entonces, cada vez que yo abro una nueva pregunta le estoy haciendo que sea más profundo el árbol, mientras más profundidad tengan significa que va a haber más separaciones.

En este caso la profundidad de este árbol es muy pequeña. Simplemente si hago dos preguntas simultáneamente: ¿es esta? Dependiendo esta respuesta me voy a la que sigue, ¿es esta? Y asigno un valor. Entonces, ahora supongamos que yo pongo un nuevo punto.

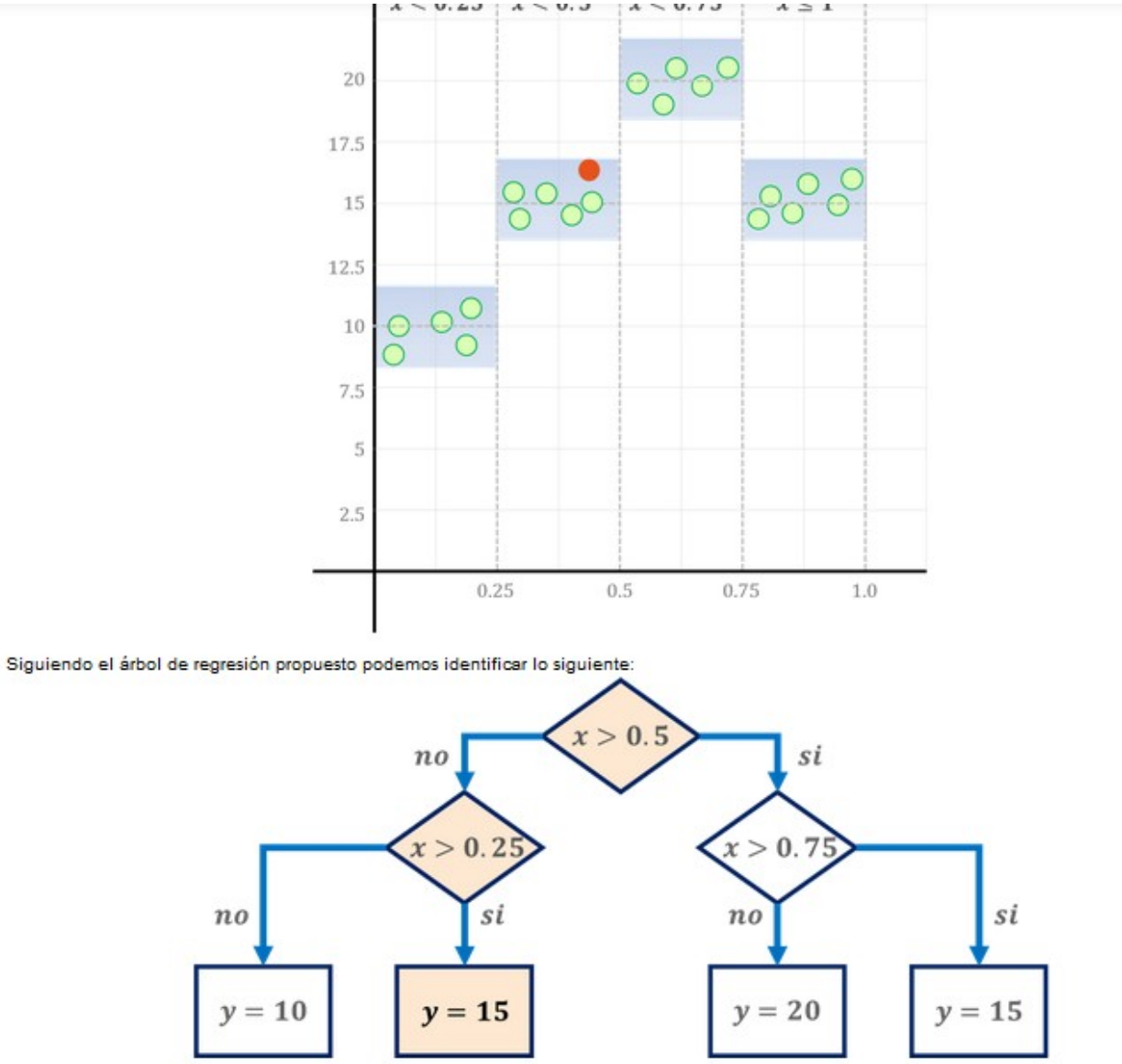


*Figura 20: Nuevo punto*

Este nuevo punto "x 0.4", no sé exactamente cuánto es su "y" porque no había tenido ese dato previamente, pero con mi árbol de regresión yo puedo preguntarme: ¿0.4 es mayor que 5? Pues no, no es mayor que 5, entonces me voy a la siguiente ¿0.4 es mayor que 0.25? Sí es. Entonces me voy... entonces inmediatamente yo sé que la "y" que le voy a asignar va a ser de 15. Así es como funciona en resumidas cuentas un árbol de regresión.

¿Cuántos intervalos vas a partir? Pues depende de la naturaleza de tus datos. ¿Cuántos valores vas a asignar? Pues un número igual a la cantidad de intervalos en las que partiste, pero eso va a generar árboles un poco más complejos en caso de que asignen muchos intervalos, o a veces insuficientes si asignas pocos intervalos. Pero bueno, más adelante se va a discutir un poquito de eso en la parte práctica.

Entonces, vamos a trabajar en conjunto con esta libreta la siguiente parte y espero que les quede esto como un apoyo y poco a poco vayan desarrollando las técnicas de árboles de regresión.

## Practicando el Árbol de Regresión

Lo que vamos a hacer aquí es utilizar las librerías que Python nos ofrece para hacer predicciones con árboles de regresión y utilizarlas en un ejemplo que ya hemos estado utilizando, que es la predicción del costo de casas. Para ello vamos a utilizar el mismo conjunto de datos y las mismas instrucciones que utilizamos en secciones anteriores. Entonces, me voy a ir un poquito más rápido en ese aspecto. Vamos a utilizar Boston, que ya incluye todo eso, todo el conjunto de datos, lo cargamos, utilizamos Pandas para hacerlo de una manera más tratable. Agregamos nuestra variable dependiente y calculamos las correlaciones.

Una vez hecho esto ya tenemos listo nuestro conjunto de datos, al menos para empezar a tratarlo. Le vamos a utilizar la misma variable que utilizamos en el ejemplo anterior, en el de SVR, llamada LSTAT. Esto va a ser la "x", y la variable "y", pues la variable la independiente, es el costo de las casas.

Entonces lo hacemos. Además, en la sección anterior, utilizamos una normalización, que es el min max que toma el rango total y lo comprime a un rango entre 0 y 1 para facilitar los cálculos internos de Python. Entonces vamos a utilizarlo: MinMaxScaler( ).

Lo importamos y hacemos una separación de los datos entre prueba y entrenamiento. Ok. Ahora sí, aquí es donde va a cambiar el cuadro con respecto al anterior, ¿por qué? Porque vamos a utilizar el árbol de regresión. Python lo ofrece a través de la librería sklearn con la librería tree. Entonces le vamos a decir: de la librería sklearn.tree importa un árbol de división pero de regresión que se llama de esta forma DecisionTreeRegressor.

DecisionTreeRegressor contiene un parámetro indispensable que es la profundidad máxima. Esto nos va a indicar qué tan amplio o qué tan profundo va a ser el árbol que vamos a generar. Esto nos va a acortar entre un árbol muy simple o un árbol muy complejo.

Vamos a utilizar uno de tamaño 2. Vamos a ver cómo se comporta. Le vamos a decir max_depth, que es profundidad máxima, igual a 2, y vamos a llamarle tree_2, nada más para identificarlos nosotros. Siempre es importante tratar de utilizar nombres de variables relevantes para lo que estamos haciendo como lo hemos estado utilizando, como x_train, por ejemplo, es el entrenamiento en "x". O por ejemplo hemos utilizado también "y", pues ya sabemos que es la variable dependiente.

En este caso vamos a utilizar tree_2, lo ejecutamos y lo tenemos listo. Falta de entrenarlo. Vamos a entrenarlo con x_train y y_train que son los que ya preparamos con anterioridad. Una vez que ya lo tenemos le vamos a decir: haz una predicción de x_train y haz una predicción de x_test.

Lo ejecutamos y guardamos las variables de respuesta. Ahora, no nos sirve de nada más tenerlo ahí y haber predicho si no podemos analizarlo de alguna forma. Vamos a crear una gráfica para poder visualizarlo.

Y aquí ya es un poco parecido al ejemplo anterior. De hecho notarán que es muy consistente los ejemplos, solamente lo que cambia es el algoritmo utilizado que es equivalente al modelo de predicción. Vamos a utilizar matplotlib, vamos a utilizar NumPy también, le decimos a Jupyter que grafique en el mismo cuaderno.

Vamos a hacer una gráfica de dispersión con "x" y "y" de entrenamiento y prueba. Este es el de prueba. Y vamos a generar muchos datos, muchos datos sobre el eje "x". Ahora, notarán que antes

nada más decíamos entre 0 y 1, y con eso nos bastaba. ¿Por qué? Porque era algo muy... muy suave la transición de la SVR, de la línea y tanto así. Como vimos en lo que nos explicaban de teoría de árbol de regresión, el árbol de regresión hace como ciertos saltos por las condiciones que están en el mismo árbol.

Entonces, vamos a generar muchos más datos para poderlo visualizar correctamente. El linspace, esta función que utilizamos para generar los datos en "x", tiene un tercer parámetro que es el número de datos que queremos generar entre 0 y 1.

Le vamos a decir que vamos a generar mil, entonces va a generar mil puntos distribuidos entre 0 y 1, es... la diferencia entre los montos es minúscula, es muy pequeña, pero nos va a ayudar a poder visualizar correctamente.

Esta línea de aquí está demás. Vamos a borrarla. Ahora sí, le decimos: el árbol... el guion bajo 2 que acabamos de generar, haz una predicción con x_plot… y lo graficamos. Aquí está. Ahora sí. Notarán que es muy diferente a los modelos anteriores, antes generábamos una línea o una curva que la subida de la línea dependía del algoritmo utilizado. En este caso utilizamos estas especies de escalones.

Estos escalones, si nos regresamos a la parte de teoría, son estos de aquí. Cada cambio que hace es una decisión en el árbol de regresión, o una condición lógica.

Entonces, estas son las hojas que está generando. Para cada una de estas líneas horizontales está generando un nodo, o una hoja terminal, que son estos de aquí. Entonces eso es lo que está haciendo nuestro árbol. Es así de simple porque le pusimos que sea de tamaño 2, profundidad máxima ,que en realidad es algo muy similar a este de aquí. Al que tenemos de ejemplo, pero ¿cómo lo podemos visualizar? O sea, si nada más lo vemos así pues no entendemos claramente el árbol, estaría más que ideal tener algo muy similar a este de aquí para poder entender lo que está haciendo nuestro modelo.

Bueno, vamos a utilizar para ello una función muy específica de Python que se llama export_graphviz. Este graphviz es un formato específico de una librería que es libre en internet en la cual nos permite generar gráficas o grafos a partir de un código establecido. Entonces, si lo ejecutamos nada más le decimos export graphviz, y como para otros pasamos nuestro modelo que queremos representar nos genera este gráfico de aquí, o este código de aquí. Intuitivamente podemos tratar de leer un poco lo que está haciendo. Nos dice que va a generar un diagrama, y nos va a generar, por ejemplo, en 0 tiene un nodo con estos datos, Tiene un nodo 1 con estos datos y empieza a hacer conexiones, de 0, 1 tiene esta conexión, pero es muy difícil de interpretarlo de esta manera.

Entonces, lo que podemos hacer es copiarlo y nos vamos a ir a una herramienta en línea. Lo vamos a buscar como graphviz online, para que esté disponible en línea, le damos click y aquí está. Nos deja un código muy similar al que tenemos nosotros. Entonces, lo vamos a pegar y nos genera este gráfico de aquí. Esto representa nuestro árbol de regresión, pero ¿como lo representa? Vamos a tratar de leerlo. Nos dice que cuando la variable independiente sea menor o igual a 0.16 se va a ir hacia la izquierda porque es un verdadero. Una vez aquí, si esa misma variable es menor o igual a 0.08, se va a ir a la izquierda y nos va a predecir con un valor de 0.77. Entonces vamos a verlo. Tomemos el valor de 0.01. Ok.

Entonces si le damos a nuestra gráfica 0.01, que sería un valor de por aquí en este rango, nos va a dar un valor de 0.77. Entonces corresponde a este primer segmento.

Tomemos otro, un número, por ejemplo, tomemos el 0.5. ¿0.5 es menor a 0.169? No, es falso. ¿Es menor a 0.39? ¿No? Ok, entonces se va a la parte de la derecha y nos va a predecir un valor de 0.2. Vamos a chequearlo. Dijimos que vamos a predecir el valor de 0.5 que está aquí y nos lo dice un valor de 0.2. O sea, nuestra gráfica corresponde al gráfico que creamos. Entonces de esta manera empieza a construir nuestro árbol. Si le vuelven a revisar el tamaño de nuestro árbol es: tamaño 1… 1 y 2. Hay dos condiciones lógicas de tamaño 2. A partir de ahí son nuestras hojas. Ok.

Así como lo hemos hecho en modelos anteriores también podemos utilizar la métrica de r2_score para calcular y comparar con los modelos anteriores, y nos da un 0.68 y 0.58 en el entrenamiento de prueba. Ahora. Bueno, ¿qué pasa si en vez de que sea de tamaño 2 lo incrementamos a tamaño 5?

Bueno, vamos a hacerlo. La única diferencia es que le vamos a decir max_depth = 5, y en vez de llamarse tree_2 vamos a llamarle tree_5. Lo ejecutamos, y lo graficamos. Ok lo que nos muestra es esta gráfica que empieza a verse, en vez de escalones claros, empiezan a dar valores arriba y abajo.

Lo que está pasando es que estamos haciendo un árbol muy muy detallado. Está acostumbrándose a estos datos que no están presentes en el comportamiento general de los datos. Entonces lo que está pasando es que estamos haciendo un overfiting, estamos sobre ajustando nuestro modelo. Es importante encontrar un balance en cuántos niveles de condición lógica vamos a generar. Nada más por graficarlo vamos a hacer este mismo diagrama de tamaño 5, y nos genera todas estas líneas, 50 líneas para un simple árbol de regresión. Si lo copiamos y lo pegamos en esta otra página que tenemos por acá pues genera un árbol muy muy grande que ya incluso se dificulta la lectura, es muy difícil tratar de ver lo que está diciendo, pero todas las condiciones lógicas que utiliza el árbol están aquí adentro.

En esta herramienta en particular aquí viene un formato, esto nos puede dar una ayuda por si queremos descargarlo, solamente le damos guardar imagen cómo, y ya tenemos nuestra imagen lista para ser utilizada. La podemos abrir. Y aquí tenemos nuestro árbol de regresión esto es en esta herramienta en particular. Ahora, Python también nos ofrece la posibilidad de, en vez de utilizar esta herramienta externa, utilizarlo internamente, pero esto ya requiere la instalación del software específico llamado graphviz, que es la herramienta que nos va a generar los árboles. Esto es... hay documentación en línea de cómo hacer esto pero para nuevos salirnos tanto del material del curso vamos a dejarlo ya a su consideración si lo quieren hacer.

Pero es importante mencionarlo que es posible. Ok. Con esto concluimos la parte práctica del árbol de regresión. En realidad es un mecanismo muy sencillo para hacer la regresión, y el árbol de regresión tiene la ventaja de que nos da un poquito de conocimiento de qué es lo que está pasando con nuestro modelo, con el SVR, por ejemplo, solamente tenemos la línea y podemos decir, pues está trabajando bien o está trabajando mal. Pero con el árbol, al darnos estas condiciones lógicas de si es menor o mayor... da estos valores...estos if... nos da un poquito de introspectiva para decir "ah...puede estar fallando porque los datos están muy cargados hacia acá". Es por decir algún, verdad.

Entonces, nos da un poquito más de conocimiento interno del problema. Una extensión natural que existe del árbol de regresión son los bosques de regresión donde utilizamos múltiples árboles de regresión y los juntamos en uno solo. De ahí de árbol a bosque. Ya hay mucha información en

internet al respecto y es una de las técnicas más utiliza das para hacer predicciones. Asimismo como SVR.

## Fundamentos de la Regresión logística

En esta sección te quiero explicar cómo hacer una regresión logística pero, sobre todo, cómo entender la teoría que se encuentra detrás para que aprendas a entender en qué escenario es aplicable y qué implicaciones tiene para el conocimiento de tu data.

Lo primero es que recordemos: la regresión logística es un modelo de aprendizaje supervisado. Eso quiere decir que busca la predicción, pero si recuerdas la regresión lineal, también es un modelo de aprendizaje supervisado que busca la predicción de una variable. ¿Cuál es la diferencia? Que la regresión lineal busca la predicción de una variable de tipo numérica, es decir, es un número. Si yo quiero predecir el número de goles que mete los equipos de fútbol en la liga española, puedo hacer una buena regresión lineal con varios valores de entrada numéricos que me permitan predecir un valor numérico.

La regresión logística comparte que es una predicción pero con la diferencia de que no va a predecir un valor numérico sino uno categórico. ¿Qué significa categórico?, que va a predecir una o más categorías, es decir, palabras. Uno puede predecir cuántos goles mete un equipo en función de variables predictivas y eso se llama regresión lineal. Pero regresión logística es predecir, por ejemplo, ¿qué equipos de basquetból de la NBA van a pasar a los playoffs? Esa predicción toma una variable de valor binomial, es decir, sí o no. Sí pasa a los playoffs o no pasan los playoffs. Eso no es un número. Y aunque tú lo conviertas en 1 y 0 no es lo más adecuado trabajarla con una regresión lineal para ese tipo de valores de salida sino que lo mejor es generar una regresión logística que trabaja en base a probabilidades.

La probabilidad de una categoría sobre la probabilidad de otra categoría. Y ese es el gran secreto de la regresión logística. Imagina, por ejemplo, que en la NBA tenemos información sobre el comportamiento de todos los equipos en todas las actividades, es decir, cuántos tiros a la canasta, cuántos tiros libres, cuántos bloqueos, cuántos minutos de juego y todos los números asociados. Y sabemos para cada equipo si pasó o no a playoffs, y con eso tenemos un valor categórico de sí o no asociado a muchos valores predictivos numéricos. Pues bien, lo que vas a ver en la computadora no es otra cosa más que aprender a construir un modelo matemático que te va permitir decir: con estas variables de entrada, ¿qué es más probable, que el equipo pase o no pase a playoffs? Y con eso puedes aplicar la matriz de confusión que ya te he explicado: a cuántos logras atinarle que pasan a playoffs y a cuántos no logras atinarle que pasan a playoffs.

No olvides que todo modelo de regresión y en general todos los modelos de aprendizaje supervisado requieren dividir la data en training set y testing set, es decir, la data de entrenamiento sobre la cual modelas y la data de pruebas sobre la cual evalúas.

## Clasificación

En secciones anteriores estuvimos viendo lo que es regresión. Ahora en estas secciones lo que vamos a ver es clasificación.

Ambos son problemas de aprendizaje supervisado. La diferencia indispensable o clara entre ambos problemas, regresión y clasificación, es que en regresión buscamos hacer una predicción de un valor

numérico, en clasificación lo que buscamos es identificar una clase: o un sí o un no, o decir: ¿es rojo? Sí o no. O ¿es perro? Sí o no. Este tipo de problemas aunque son similares a la regresión tienen una característica muy especial que hay que tener principal cuidado con cómo vamos a manejar los datos de salida.

Para ello vamos a utilizar el cuaderno en Python llamado "regresión logística".Una vez que ya abrió vamos a leerlo un poco. Dice: la regresión logística, o clasificación, es un algoritmo de clasificación que es utilizado para predecir la probabilidad de que un evento un dato pertenezca a una categoría. En particular la clasificación, la variable de salida es una variable binaria: 0, 1, verdadero o falso. Para llevar este ejemplo a la práctica vamos a utilizar este conjunto de datos que tenemos aquí. Vamos a darle clic, y este conjunto de datos es referente a un banco. En ese conjunto de datos lo que hacen es... tienen datos de usuarios y la variable de predicción es... a partir de una llamada que les hacen les ofrecen una inversión a corto plazo la persona lo acepta o lo rechaza.

Entonces es muy importante identificar cuáles personas van a aceptar ese recurso o esa inversión. ¿Por qué?. Porque no queremos llamarle a todo y molestarlos. Solamente queremos marcarle a las personas que sí es probable que lo acepten. Por eso es muy importante clasificar entre las personas cuáles es probable que lo acepten y cuáles no. En este caso tenemos que el conjunto de datos tiene múltiples variables. Algunas son reales, otras categóricas, pero la variable de respuesta es una variable "si, no", es binaria. Entonces corresponde con un problema de clasificación. Vamos a descargarlo. Le pueden dar clic aquí donde dice "data folder", y van descargar este que se llama "bank.zip", y lo a empezar a descargar. Vamos a abrirlo... en mostrar carpeta y aquí lo tenemos. Vamos a descomprimirlo, extraer, extraer. Voy a cerrar esto de aquí… y ya tenemos nuestra carpeta con los archivos necesarios. Esto lo van a copiar a su carpeta donde tienen los cuadernos de Python, ya lo hemos hecho con anterioridad, ya lo tengo descargado en este caso. Entonces, voy a cerrar esta parte aquí.

Ok. Una vez que ya los tenemos simplemente hay que cargarlos a Pandas. La particularidad de este archivo de texto, o de este archivo csv, es que los datos no están separados por comas, sino por puntos y comas. Es algo muy particular de este documento pero Pandas lo puede tratar simplemente agregando ese parámetro.

Entonces los leemos y ya nos podemos hacer una idea de cuáles son los datos que incluye este conjunto de datos: tenemos, por ejemplo, la edad de la persona, su trabajo, su estado civil, su nivel de educación, la duración de su contrato, el balance actual de su cuenta, entre otras cosas. Una vez que cargamos nuestro conjunto de datos es algo indispensable en los programas de clasificación determinar el balance de la variable de respuesta. ¿Qué es eso? Queremos decir cuántas personas han dicho que no y cuáles han dicho que sí.

Si está muy cargado, por ejemplo, a no. Es muy probable que el algoritmo lo que haga es simplemente predecir siempre que no. ¿Por qué? porque pues es más probable que si nada más dice "no" pues se equivoque en muy poquita proporción la que es "sí". Aquí lo podemos visualizar. Vamos a verificarlo con una gráfica de barras, y le vamos a decir que en el eje de las "x" grafique los valores únicos, es decir "no" y "sí", y en el eje de las "y" imprima los conteos de esos datos. Entonces tenemos que alrededor de 4000 personas dicen que no en ese conjunto datos y solamente alrededor de 500 dicen que sí.

Entonces, está muy desproporcionado. Hay que hacer algo para poder arreglarlo. Para ello tenemos el resampling. Resampling lo que nos va a decir, o lo que nos va a ayudar es poder equilibrar los

conjuntos de datos ya sea quitándole a la variable que son mucha información o a la que es la variable que tiene pocos agregandole información. Esto se llama undersampling o oversampling, o submuestreo, o sobremuestreo.

Entonces, el submuestreo, que es la parte de la izquierda, lo que hace es: toma... en este caso específico toma los registros que dicen "no" y los corta, ¿para qué? Para que queden del mismo tamaño que los que... en el caso de decir sí. El oversampling lo que haría es tomar la parte donde las personas dicen que sí y empezarlas a duplicar de forma de que llegue el mismo número de personas. Ok.

La manera más fácil de hacerlo es separar el conjunto datos original entre "no" y "sí", y ya sea cortar o duplicar. En este caso vamos a utilizar el undersampling, es decir, vamos a cortar. Entonces, para ello vamos a utilizar esta parte de aquí que es: del conjunto de datos original dame aquellos donde la variable de respuestas sea "no", y guárdalos con "df_no".

Algo similar al "no" con la parte de "sí". Ejecutamos y ya tenemos esos dos conjuntos de datos separados. Podemos ver la forma de estos datos, y es de tamaño 4000 contra 521. Sigue estando muy desbalanceado. Entonces, este conjunto de datos... el "no" hay que cortarlo, pero no lo vamos a cortar nada más así, o sea, hay que hacer un muestreo de esos datos para que no se quede muy cargado al principio del conjunto de datos o al final el conjunto datos, sino de una forma aleatoria. Entonces le vamos a decir al "df_no": dame una muestra de tamaño 521. ¿Por qué 521? Porque es el tamaño de nuestro conjunto de datos "sí".

Le podemos decir: dame la forma, y ya tiene exactamente la misma forma que nuestro conjunto de datos "sí". Ahora, ya los tenemos del mismo tamaño, hay que recombinarlos para que quede como un solo conjunto de datos. Pandas nos ayuda en esta tarea, solamente le vamos a decir: Pandas, concatena estos dos conjuntos de datos, que es el df_no, reducido, es el que trabajamos aquí arriba, y el "df sí" (df_yes), que es el que ya teníamos original, y lo vamos a hacer no en el eje de las columnas sino en el eje de las filas para que queden uno arriba del otro. Si lo hacemos e imprimimos la cabecera y la cola de este conjunto de datos nos queda amontonado en la parte de arriba los "no" y en la parte de abajo los "sí". Ya podemos utilizar este conjunto de datos pero lo conveniente sería revolverlos para que no queden simplemente encimados, sino queden revueltos. Entonces los vamos a revolver a través de sample. Sample en vez de pasar un número específico, le vamos a pasar una fracción, un porcentaje. Si le pasamos el porcentaje de uno le estamos diciendo: en vez de darme un muestreo de de una parte reducida dame un muestreo de todo. Entonces, efectivamente lo que va a hacer es revolverlos, no muestrear, revolver. Entonces si lo ejecutamos nos da el conjunto de datos ya reducido y revuelto listo para ser utilizado.

Lo podemos comprobar la forma que nos da 1,042 filas contra 17 columnas. Entonces está perfecto para empezar a utilizarlo. Ahora sí ya podemos hacer lo que hemos estado utilizando con anterioridad. Hay que seleccionar una variable a la cual vamos a utilizar como variable independiente que en este caso vamos utilizar la duración. Una vez terminado el ejercicio los invito a cambiarlo por... no sé... tal vez el balance o la edad, y eso lo vamos guardar como "X" y como "y". De forma similar a los ejemplos de regresión vamos a hacer un split de los datos entre entrenamiento y prueba.

Una vez que termine de ejecutar vamos a chequear las formas, y ahora sí, tenemos que en "X" de entrenamiento va a utilizar un conjunto de datos de tamaño casi 700, 698, y como prueba vamos a

utilizar 344 datos. Ok. Ahora, ya tenemos todos los datos listos, ya podemos empezar a hacer predicciones.

Para ello dijimos que vamos a utilizar una regresión logística que simplemente importamos, gracias a Python, de esta manera le decimos: sk-learn, de la parte de modelos lineales, importa regresión logística.

Lo que va a hacer regresión logística es funcionar exactamente igual a los modelos de regresión. Eso no va a cambiar. Les repito que el patrón de cómo utilizamos los modelos en Python es muy similar. Entonces, si ya conocemos cómo hacer regresiones, por ejemplo, ya podemos hacer clasificaciones directamente. Incluso podemos reutilizar los algoritmos que hemos utilizado con anterioridad, ya sea árboles de regresión, SVR o regresiones lineales, pero en su versión de clasificación.

Todos los algoritmos tiene una versión de clasificación. Entonces, solamente vamos a cambiar la librería y el conjunto de datos, verdad. Ahora sí. Vamos a continuar en regresión logística, vamos a utilizar esta parte de aquí, vamos a especificarle un solver, ejecutamos solamente así, por el momento nos da un warning, y nos dice que el solver default va a ser configurado a "lbfgs" que es el motor de cómo resuelve el problema Python internamente, entonces para que no salga esta notificación pues simplemente lo agregamos, y eso nos permite dejar este código listo para el futuro. Tal vez a ustedes no les salga este código, tal vez lo están haciendo en un futuro donde este warning ya no existe. Pero por el momento lo tenemos que poner. Ok.

Si lo ejecutamos, para este momento ya usamos la función fit, para entrenar con X_train y y_train, hicimos la predicción con X_train, e hicimos la predicción con X_test, y le decimos: "clf", dame un score, clf es nuestro modelo que acabamos de crear acá arriba, y el score lo podemos obtener con X_train y y_train, también con X_test y y_test, como está aquí abajo, pero el principio dijimos que nos daba las probabilidades. Para obtener probabilidades lo que podemos hacer es utilizar predict_proba, es una función muy específica de regresión logística que vamos a imprimir… predict_proba... lo que nos va a dar esta función es, vamos a imprimirlo aquí a su primer lote por un montón de probabilidades un conjunto grande de probabilidad el cual nos dice para que uno de los elementos que le pasamos esto es la probabilidad de que sea uno esto es para el segundo conjunto de datos que se prevé que sea uno pero es gráficamente lo ponemos de esta manera ya tenemos una gráfica que está muy cargada a valores grandes o probabilidades grandes cercano al cero y cuando incrementa las probabilidades son cercanas a cero hasta el momento no hemos explicado por qué se llama regresión logística si notan esta gráfica que tenemos aquí pareciera que está formando una especie de curva de s en estadística y matemáticas en la función llamada función logística y se observa una gráfica de aquí es muy similar a la gráfica que tenemos cuando de python es una función ya establecida que uno sobre bueno más o menos t este es el evento y lo que está haciendo es decir para la variable independiente que estamos utilizando esto es la probabilidad de que el evento ocurra y si por ejemplo para un valor de x por ejemplo digamos 100 tenemos un valor de punto 7 pues eso hay una prioridad de punto 7 de que sea un no eso lo que no está representando nuestra gráfica de aquí la reversión logística es una herramienta muy importante en lo que es machine learning porque es una de las principales herramientas para poder generar una clasificación de la forma en la que lo tenemos ahora no nos está diciendo mucho si está comportándose correctamente o está muy equivocado, solamente tenemos la gráfica no tenemos una forma de evaluarlo correctamente.

Entonces una de las alternativas que hay para poderlo evaluar eso utilizamos la matriz de confusión esto lo vamos a ver en la siguiente sección y nos va a permitir conseguir un poquito más de conocimiento de cómo está funcionando nuestra reclasificación.

## Introducción a la Matriz de Confusión

En esta sección quiero explicarte qué es la matriz de confusión y lo más importante cómo la puedes interpretar. Uno de los grandes retos que vas a abordar en este curso es que no solamente se trata de generar algunos algoritmos o procesos que te permitan conocer la información, sino que tienes que saber evaluar el nivel que tiene tu modelo y la evaluación de las regresiones lineales, los modelos de regresión logística, los árboles de decisión.

Todos ellos pasan por una necesaria matriz de confusión que aparte es muy útil en diversas ciencias y te será bastante interesante aprender a interpretarla. Muy bien, te explico. La matriz de confusión es en primer instancia una matriz, eso quiere decir que, pon atención, tiene dos cosas: tiene filas y columnas. La primera fila, como verás, dice "PP". ¿Qué significa? Predicted Positive, es decir, Predicción Positiva y la segunda fila dice Predicted False que significa Predicción Falsa.

Entonces, fíjate, primera fila son todas las predicciones que nuestro modelo dice que van a ser positivas y la segunda fila son todas las predicciones que nuestro modelo dice que van a ser falsas. Intentemos entenderlo en el marco de un ejemplo un poco aterrizado. Imagina que estamos haciendo un diagnóstico para saber qué tipo de pacientes podríamos hacer una predicción de si presentan cáncer o presentan algún tipo de padecimiento.

Todos los Predicted Positive son los que nuestro modelo dice: "este paciente, de acuerdo a los datos estudiados, va a tener cáncer". Ese es un Predictive Positive. Ahora, qué pasa si el modelo me dice: "este paciente según los datos estudiantes no tendrá cáncer". Ese es un Predictive False. Ahora, ¿dónde viene lo bueno? Que un Predictive Positive, para que sea valioso en nuestro modelo, tiene que compaginar con los True Positive, es decir, los Verdaderos Positivos. De todas las personas que yo digo en mi modelo que van a tener cáncer, ¿cuántas verdaderamente en el data de prueba tienen cáncer? Y vienen nuestras columnas a entrar en acción.

La primera columna se llama True Positive y la segunda columna True False. Aquí están los que yo digo, más bien, mi modelo dice que serán positivos y aquí los que verdaderamente son positivos. Entonces tengo que de los que yo dije que iban a ser positivos, son 85 y 5; en realidad son 90, pero de los 90 que predije positivos, ¿cuántos crees que fueron verdaderamente positivos? Donde intersectan mi columna con mi fila, es decir, 85 de mi data de prueba. ¿Cuantos de los que yo predije positivos son verdaderamente falsos? 5.

Eso es importante, quiere decir que le estoy atinando, por ocupar esa palabra a los que están en la diagonal mayor de mi matriz: 85 de los 90 predicciones que hice fueron verdaderamente positivas y 5 falsas; y la fila de abajo es lo mismo: De los pacientes que yo dije "no van a tener cáncer" la data me dice que debo diagnosticarlos como que no, ¿A cuántos verdaderamente le atiné? Pues a los True False, a los verdaderos falsos.

Yo dije, "era falso" y ¿cuántos sí fueron falsos? 45. ¿Cuántos dije en total que iba a predecir? Yo predije 45 más 15, 60. De los 60 ¿Cuántos le atiné? A 45. Y ésta es la matriz de confusión es muy sencillo de aquí sacas dos términos importantes: el Class Precision y el Class Recall, que quiere decir qué tan buena es la predicción y qué tanto abarca de los verdaderos positivos. Vayamos

primero horizontalmente. Yo hice una predicción de 90. De esas predicciones de 90 ¿Qué tan eficiente fue? Es decir, ¿Qué tanto me equivoqué? Pues me equivoqué en 5 de 90 si tú sacas el cálculo de 85 con respecto a 90 el porcentaje de acierto es 94%. Eso se llama Class Precision: Qué tan bien hice que mi clase fuera precisa para atinar a los elementos que quiero predecir. Predicted False es cuántos ahora agarre en porcentaje de mi total de predicción. Yo predije para 60 pero le atiné sólo a 45, por lo que el porcentaje de precisión de clase baja a 75%.

Y ahora tenemos las columnas. Estos de aquí abajo se llaman Class Recalls que quiere decir: cuántos de los que eran totales verdaderamente positivos logré captar, es decir, en este caso, tengo 85 más 15 que son 100 personas de toda mi data que sí tienen cáncer. Estas personas que están aquí, esta suma, 85 más 15, es la gente que verdaderamente tiene cáncer. Cuánto de esa población logré captar correctamente con mis predicciones. Aquí implica que cuántas están en mi diagonal mayor: agarré 85 de 100, 85%. En mi Class Recall segunda: de los 50 verdaderos falsos que hay en todo mi data set, ¿Cuántos de esos verdaderamente los agarré como falsos? Están en mi diagonal mayor: 90%. Esto nos ayuda a entender qué tan bueno es el modelo o no. Ahora lo ocupe con cáncer que es un poco dramático, pero se puede utilizar por ejemplo en la regresión logística para saber qué equipos de fútbol o de básquetbol van a pasar a playoffs.

Y tú lo puedes aplicar, de hecho si viste la película de Moneyball es justamente en lo que consiste. Y la regresión logística, al ser un cálculo de probabilidad, pues te va a decir cuántos caen en cada cuadrito y tú podrías decir "ah bueno mi modelo es mejor para predecir los verdaderos positivos que los verdaderos negativos, tengo mejor precisión", y también podría decir, "pero capto mejor los falsos". ¿Se fijan esa diferencia? Depende el enfoque de tu problema. Puede que tu problema esté centrado en encontrar con precisión la clase que predices. Eres buenísimo para predecir cuándo tienen cáncer porque no fallas en tu predicción pero, ¿A quiénes captas más?, ¿a los verdaderos o a los falsos? Captas más a los falsos y esto te plantea todo el escenario.

Piensa en el problema del cáncer, ¿Qué es más peligroso, un error tipo falso positivo o un error tipo falso negativo? ¿Qué es más grave, que yo le diga a un paciente usted tiene cáncer y no tenga o que le diga usted no tiene cáncer y sí tenga? Depende del enfoque que planteé tu problema es el análisis que debes hacer con tu matriz de confusión.

## Matriz de Confusión

En la sección anterior ya vimos lo que es una clasificación y cómo poderla hacer a través de Python. Lo que vamos a hacer en este caso es utilizar una matriz de confusión para poder valorar nuestro modelo y ver qué tan bien se comportan.

Matriz de confusión es en realidad una de las alternativas que hay para poder evaluar un modelo, pero es uno de los más utilizados. Una matriz de confusión es una herramienta que nos permite visualizar y comprender de una manera rápida el comportamiento de un modelo de clasificación. Su principal ventaja, además de decirnos qué también se comportó el modelo, es qué tipo de errores cometió.

Una matriz de confusión generalmente tiene esta forma que tenemos en el cuaderno, que es en las filas las predicciones, ya sean positivas y negativas, y lo real o lo actual, que es positivos y negativos.

Aquí mismo hay un cuadro que tiene estas siglas: TP, FP, FM y TN. ¿Qué representan estos? Bueno, el TP, que es el primer cuadrado, que es donde se encuentran positivo de predicción y positivo de actual, es un verdadero positivo. Aquí caen todas las predicciones que hagamos... que predijimos como positivas y realmente son positivas. En la esquina contraria tenemos TN que es un verdadero negativo, es decir, todas las proyecciones que hicimos como "no" y que en realidad son "no". Para que nuestro modelo sea, digamos, perfecto las proyecciones que hagamos deben de caer en estos dos cuadrados: en TP y TN. Desafortunadamente el mundo no es tan sencillo y en la naturaleza, en los problemas reales, cometemos errores ya sea tanto FPs o FNs. ¿Qué representan? FP es cuando hacemos una predicción y decimos que es verdadero pero en realidad es falso, es un falso positivo, y esto se conoce como un error del tipo 1.

También tenemos el contrario: cuando hacemos una predicción y decimos que es un "no" pero en realidad es un "sí", esto es un falso negativo y es un error de tipo 2.

Bueno, ¿pero esto cómo lo implementamos en nuestro código para nuestro modelo ya hecho? Para ello vamos a utilizar Python. Python ya nos ayuda con una matriz de confusión, así se llama, confusion matrix. Para ello lo utilizamos bajo la librería sk-learn dentro de metrix o métricas, y en este caso hay que especificarle las etiquetas que utilizamos para clasificar verdadero o falso.

En nuestro conjunto de datos que tenemos por acá arriba ya sabemos que es la palabra en inglés para "no" y para "si": "no" y "yes". Entonces, vamos a utilizarlas. Vamos a decirle que las etiquetas son "yes" y "no", lo guardamos en una lista y utilizamos la matriz de confusión. Ahora, aquí no necesitamos pasarle el modelo el arreglo que tenemos que pasarle es las etiquetas reales y las etiquetas predichas. A partir de eso Python pueden hacer toda las matriz de confusión y darnos los datos necesarios.

Entonces, le pasamos como primer parámetro las "y" reales, y como segundo las predicciones. Como tercero le pasamos los etiquetas que acabamos de generar y nos da esta matriz. Lo que representa es que el primer número de la matriz es el TP, la última es el TN que son los valores que estamos buscando que están muy altos y todos los errores son este de aquí, el 60 que es un FN, y el 129 que es un FP. Ok. Esto es con el entrenamiento.

Generalmente no queremos conocer el error de entrenamiento, queremos conocer el error de validación, es el que nos interesa, ¿cómo se va a comportar en el mundo real? Para ello vamos a utilizar en lugar del de entrenamiento el conjunto de pruebas. Además le vamos a decir a Python que no nos muestre una matriz en forma de texto, como aquí arriba, lo queremos de una forma agradable a la vista. Para ello vamos a utilizar un mapa de calor que nos ofrece la librería sns, o seaborn. Vamos a decirle que genere una matriz de confusión con el conjunto de prueba, con las etiquetas que ya establecimos y lo vamos a guardar como cm. Cm refiriéndose a matriz de confusión en inglés.

Vamos a decirle a Python que haga una gráfica de calor con los datos que acabamos de generar, queremos que haga anotaciones, es decir, que ponga el valor numérico en cada una de las celdas... le estamos diciendo que esas anotaciones las queremos en dígitos de forma que no pongan números decimales que afecten la visualización. Le decimos también que las marcas que va a poner en "x" y "y" son las mismas de las etiquetas y...por último, que los colores que utilice sean verdes para poder determinar de una forma agradable qué tan bien o qué tan mal le fue al modelo de clasificación.

Entonces, lo imprimimos y nos ofrece esta matriz de aquí que ya puede parecerse un poquito más a la que tenemos de ejemplo. A partir de aquí ya podemos identificar nuestros TP, nuestros TN, nuestros FP y nuestros FN.

Ok, ya sabemos que TP es la primera celda de aquí. Sabemos que TN es la última. Sabemos que el FP es esta de aquí, la segunda y FN es la tercera. Ahora, en vez de estarlos guardando uno por uno estos valores, lo que podemos decir es: cm, que es la matriz de confusión, dame la versión lisa de esa matriz, y esto nos va a dar 103, 65, 31 y 145. Python nos permite, a partir de una colección de datos o una lista, guardar múltiples variables al mismo tiempo. Entonces, si respetamos el mismo orden, el TP, FP, FN y TN, podemos guardar directamente esos datos dentro de las variables respectivas. Entonces, vamos a hacerlo, y si imprimimos TP no da un valor de 103. Si imprimimos, por ejemplo, TN nos da un valor de 145. Entonces, ya tenemos los valores de la matriz directamente en variables.

¿Para qué queremos esto? Bueno, hay algunas funciones o algunas fórmulas que podemos utilizar para poder determinar qué tan bien o qué tan mal le fue a nuestro modelo. Estas funciones ya están preestablecidas pero las vamos a ver aquí ligeramente. La primera de ellas, vital para este tipo de cálculos, es el recall, que es el porcentaje de casos positivos correctamente clasificados. Este se maneja con las siglas TPR, y esto es igual al TP, que son los true positive, o sea, los casos donde se predijo como correcto y están correctos, sobre TP más FN, es decir, esta columna de aquí... sobre todos los datos donde debía de ser verdadero. Eso nos da el recall.

Lo podemos ejecutar así directamente. Le decimos: TP sobre la suma de esos dos valores, y eso nos da el TPR. Y nos da 0.76.

Otro valor importante es la precisión, o "precision", que es el porcentaje de los que realmente son correctos a partir de los casos clasificados como correctos. Es una fórmula muy similar pero en vez de considerar la columna, considera el renglón. Entonces, lo hacemos de una forma muy similar y nos da ese valor de 0.61. Ahora ¿qué es lo que quieren representar estas variables? Bueno, por aquí lo tenemos descrito. Un recall alto con una precisión baja indica que la mayoría de los casos positivos son clasificados correctamente, pero hay muchos falsos positivos.

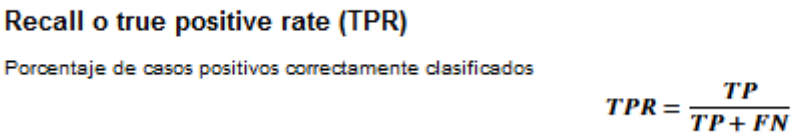


*Figura 21: True positive rate*

El contrario, es decir, un TPR bajo con un PPV alto, indica que la predicción falló a muchos casos positivos, pero los que clasificamos como positivos muy probablemente estén correctos.

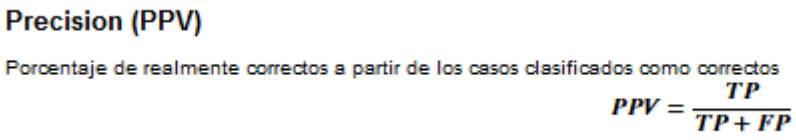


*Figura 22: Precisión*

Otra de las fórmulas que tenemos es la exactitud que manejamos con las siglas ACC. Esto lo que nos quiere dar es el número de veces que predijo correctamente tanto positivos como negativas sobre el total de los datos. Entonces, es lo que tenemos aquí: TP más TM, o sea, los "trues", sobre la totalidad de los datos. Lo podemos ejecutar de esa manera directamente TP más TN sobre la suma de todo. Y nos da 0.72.


*Figura 23: Accurracy*

Por último tenemos el F-measure, o la medida f, que es la media armónica entre el TPR y el PPV. ¿Por qué la media armónica y no la media aritmética? Bueno, la media armónica tiende a penalizar más si uno de los dos valores es muy alto o muy bajo. Lo que busca es que estén similares o parecidos, y esta media armónica se calcula de esta manera... que tenemos dos veces el TPR por PPV, sobre a suma de esos dos mismos valores, y simplemente lo imprimimos, 0.68. Todas estas medidas nos dan alguna forma de poder valorar nuestro modelo ya en una forma numérica como lo hacíamos con el r2 score en la regresión.


*Figura 24: Medida de F*

No tenemos que estar haciendo todas estas fórmulas cada que vayamos a hacer nuestro modelo. Python afortunadamente ya puede hacer muchos de estos cálculos por sí solo. Para ello utilizamos el classification report ya dentro de sk-learn, simplemente le pasamos los conjuntos de prueba de "y" y de predicción de "y".

Asimismo también tenemos esta de aquí que es la medida de exactitud que ya está implementada dentro Python, y los podemos ejecutar, y los va a generar sin ningún problema. Por ejemplo, de exactitud nos da un valor de 0.72 si subimos aquí tenemos un valor de 0.72, coinciden. ¿Por qué? porque ya nosotros calculamos los valores directamente de la matriz de confusión.

Además tenemos en el reporte de clasificación los valores de precisión, de recall y de f-score, tenemos una precisión de 0.77 y un recall de 0.61 que coincide con lo que tenemos: 0.768 y 0.61, nada más que están redondeados, y el f-score es de 0.68 y 0.68. Entonces, todas estas métricas que nos da Python ya están calculadas, no tenemos que estar haciendo fórmulas nosotros manualmente pero es bueno saber de dónde vienen.

Esta es una de las formas de poder evaluar nuestro modelo de clasificación. Otra alternativa es utilizar las curvas ROC que es lo que veremos en una sección posterior. Pero por el momento espero que les haya quedado claro cómo funciona un modelo de clasificación y cómo tratar de evaluarlo.

# Semana 3

## Introducción

En la semana anterior realizamos diversas estimaciones con múltiples modelos como regresiones lineales, SVR, y árboles de regresión. Esta semana la arrancaremos con la pregunta: ¿Cómo podemos mejorar estos modelos ya creados?

Con ello te darás cuenta de que esta semana es complementaria a la semana anterior. Arrancaremos con explicar maneras más gráficas de caracterizar un modelo de clasificación, además, aprovecharemos la información contenida en las variables categóricas que hasta el momento han sido ignoradas.

Normalizaremos los datos y ejecutaremos métodos automáticos de selección de variables, como siempre; haremos estas tareas con ayuda de nuestros cuadros de Jupyter. En esta semana aparecerán muchos nuevos términos, como: Curvas ROC, Curvas CAP, Variables Dummy, entre otros.

## Curvas ROC

En semanas anteriores vimos cómo crear modelos predictivos y poder analizarlos y medirlos para ver qué tan efectivos son. A veces estos números que son arrojados en la evaluación de nuestro modelo no son suficientes para darnos la idea si funcionan bien o mal. Tal vez tiene un 0.7, pero pues eso no nos dice si le fue muy bien o muy mal. Falta contextualizar esa idea en nuestras cabezas.

Entonces, para ello hay algunas técnicas que nos permiten mejorar esta interpretación de los modelos a través de algunas curvas. Estas son llamadas curvas ROC y curvas CAP. En este cuaderno vamos a empezar a analizarlas y ver cómo se van construyendo paso a paso. Para ello vamos a utilizar un cuaderno, que es el cuadro lo que estaremos utilizando a lo largo de esta semana. Este se llama mejorando los modelos.

Una vez que hayan abierto el cuaderno les va a aparecer el siguiente texto. Primero hay que clarificar con qué conjunto de datos vamos a estar trabajando, y es el mismo de la semana pasada, que es el que tenemos aquí. Si recordamos utilizamos el Bank Marketing Data Set, que es un conjunto de datos de un banco y que en este banco lo que tratan de hacer es ofrecer inversiones a corto plazo. El conjunto de datos es multivariado, tiene diferentes tipos de variables, tiene atributos reales y es un problema de clasificación. ¿Por qué? Porque queremos predecir si el cliente va a aceptar o no esta inversión a corto plazo.

Entonces, ese conjunto de datos lo podemos descargar directamente de aquí : Data Folder y "bank.zip", es un archivo comprimido, o ese conjunto datos ya debe estar disponible dentro de su conjunto de datos. Una vez ya hayan descargado o colocado los datos donde corresponde vamos a importarlos con Pandas para empezar a analizar nuestro conjunto datos que ya hemos utilizado, entonces lo vamos a ver más rápidamente. Para ello vamos a utilizar, como siempre, como lo hemos estado haciendo, Pandas.

Vamos a hacerle el import, "import pandas as pd", aparte vamos a utilizar NumPy para todas las operaciones de arreglos que podamos utilizar, y de una vez, porque sabemos que lo vamos a utilizar, vamos a importar la librería matplotlib que nos va ayudar a graficar.

Nuestro conjunto de datos ya se encuentra en nuestra carpeta "Data" en una carpeta interna que se llama "Bank", y el archivo que vamos a analizar se llama "bank.csv". Este archivo tiene la particularidad de que no está separado por comas, sino por puntos y comas. Una vez que ya lo hemos procesado vamos a transformar la columna "y". Si checa en el archivo original la columna "y" tiene etiquetas de sí o no, dependiendo si el cliente aceptó o rechazó esta inversión a corto plazo. Vamos a cambiarlo por valores numéricos para facilitarnos el trabajo más adelante. Aquí mismo le vamos a decir: en la columna "y" reemplaza los "sí" por 1's y reemplaza los "no" por 0's y guardarlo en la misma columna "y".

Una vez hecho eso lo podemos imprimir y ya vemos que ya tiene el conjunto de datos ya preparado, y nuestra columna "y" ya está limpia. Una vez que ya tenemos el conjunto de datos preparado vamos a tratar de equilibrar las diferentes clases de nuestro conjunto de dato. Lo vimos en la semana anterior pero esto es una técnica de undersampling la que vamos a utilizar. Lo que queremos hacer es que se equilibre los "sí" y los "no" en nuestro conjunto. Entonces, para ello vamos a separarlo en dos conjuntos diferentes. Todos los clientes que dijeron "sí" y todos los clientes que dijeron "no". Vamos a reducir el más grande de ellos que ya sabíamos que había más personas que decían "no" que personas que decían que "sí".

Es más probable que te rechacen en este conjunto de datos. Entonces, el "no" lo vamos a reducir. Vamos a hacer un muestreo de estos datos, vamos a tomar una muestra, ¿de qué tamaño? Del tamaño del conjunto de datos "sí". Ok. Esta variable aquí df_no_reduced nos va a contener un conjunto de datos del mismo tamaño que df_yes. Ahora nos queda unirlas para poder tener un conjunto de datos ya completo, y esto lo vamos a hacer a través de la función "concat". Vamos a concatenar los "no" con los "sí" a través del eje 0, es decir, a través de las filas, se pone uno arriba del otro.

Y para que no queden un conjunto arriba que diga no y todo el conjunto de abajo que diga sí, vamos a revolverlos, y para esto vamos a hacer otra vez un muestreo, pero con este parámetro, "fraction", "frac=1" para que sea del mismo tamaño, es decir, sólo se revuelve. Una vez hecho eso ya tenemos nuestro conjunto de datos ya revuelto con 0's y 1's, y el número de 0's y el número de 1's debe ser equivalente. Esto lo podemos validar con la función "value_counts" sobre la columna "y".

Entonces, ya nos dice que hay 521 1's y 521 0's. Ahora sí, ya podemos empezar a dividir nuestros datos para poder empezar a entrenar nuestros modelos. En nuestro conjunto de datos ahora falta dividirlo en variable "x" y variable "y". Variable "x" son nuestras variables independientes y "y" nuestra dependiente. Por lo tanto vamos a utilizar nada más dos columnas para empezar a simplificar la tarea, vamos a utilizar duración y edad en nuestra variable "x". Como variable "y" pues solamente tenemos la variable respuesta que, convenientemente en este conjunto datos, se llama "y"

A partir de ahí vamos a hacer un split de los datos, justo como lo hicimos la semana anterior. Vamos a separar X_train, X_test, y_train y y_test. Esto con la función train_test_split que ya Python nos proporciona para poder hacer la separación de los datos. Ahora sí, en este momento ya tenemos nuestro conjunto datos preparados. Ahora vamos a hacer una regresión. En específico vamos a una regresión logística, vamos a utilizar LogisticRegression, y lo vamos a entrenar con los datos de entrenamiento. Recuerden, no podemos utilizar nada de prueba en el entrenamiento. Entonces, vamos a utilizar X_train y y_train. Eso nos va a generar un modelo el cual le vamos a llamar "clf",

con "clf" vamos a predecir, ¿qué? Pues entrenamiento y prueba para ver cómo se contrarresta uno con el otro.

Algunos de ustedes podrían preguntarse por qué utilizamos la variable clf. Clf es una variable que se utiliza como convención con la librería sk-learn que está incluida dentro de Python, y es la que nos ayude a crear nuestro modelos, entonces, para no romper con ese estándar o esa tradición vamos a seguir utilizando esa variable. Ok.

Una vez que ya tengamos nuestro modelo entrenado falta validarlo. Para ello vamos a utilizar classification_report y esta métrica de precisión., las imprimimos y nos da un valor de 0.72. Ok. Eso es lo vimos hasta el momento de la semana pasada, ahora, ¿cómo lo puedo visualizar de alguna otra manera? Bueno, para ello vamos a utilizar primeramente la curva ROC. La curva ROC, o la curva de característica operativa del receptor, que es un nombre medio raro, por lo tanto vamos a utilizar ROC simplemente, es un método adicional para medir la calidad de un modelo de clasificación binario. Precisamente lo que hemos estado haciendo. Para graficarlo en realidad es muy sencillo, simplemente se contrasta lo que es el TPR contra el FPR, que lo vimos en la semana anterior.

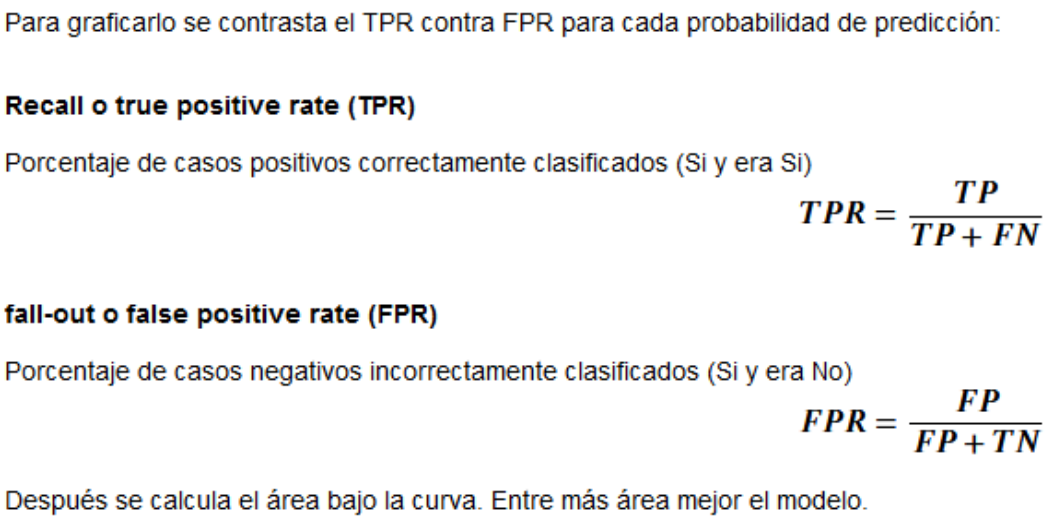


*Figura 25: Cálculo de TPR y FPR*

Y lo que va a generar esta curva es... en realidad va a generar una curva, lo que nos va a decir esa curva es que entre más área bajo la curva, mejor el modelo. Vamos a visualizarlo. Para ello Python ya tiene esta curva programada. Vamos a utilizar lo que es la roc_curve, y vamos a utilizar esta otra librería que se llama roc, precisión, score, o auc de precisión (roc_auc_score). Vamos a, primero, hacer una figura, vamos a agrandarla un poquito para que sea 10 con 6, y vamos a graficar una línea roja desde el 0, 1 hasta el 1,1, y lo que nos va a marcar esa línea es el comportamiento promedio o aleatorio.

```python
from sklearn.metrics import roc_curve, roc_auc_score

plt.figure(figsize = (10, 6))
plt.plot([0,1], [0,1], 'r--')

probs = clf.predict_proba(X_test)

probs = probs[:, 1]
fpr, tpr, thresholds = roc_curve(y_test,probs)
roc_auc = roc_auc_score(y_test, y_test_hat)

label = 'AUC:' + ' {0:.2f}'.format(roc_auc)
plt.plot(fpr, tpr, c = 'g', label = label, linewidth = 4)
plt.xlabel('FPR', fontsize = 16)
plt.ylabel('TPR', fontsize = 16)
plt.title('ROC', fontsize = 16)
plt.legend();
```

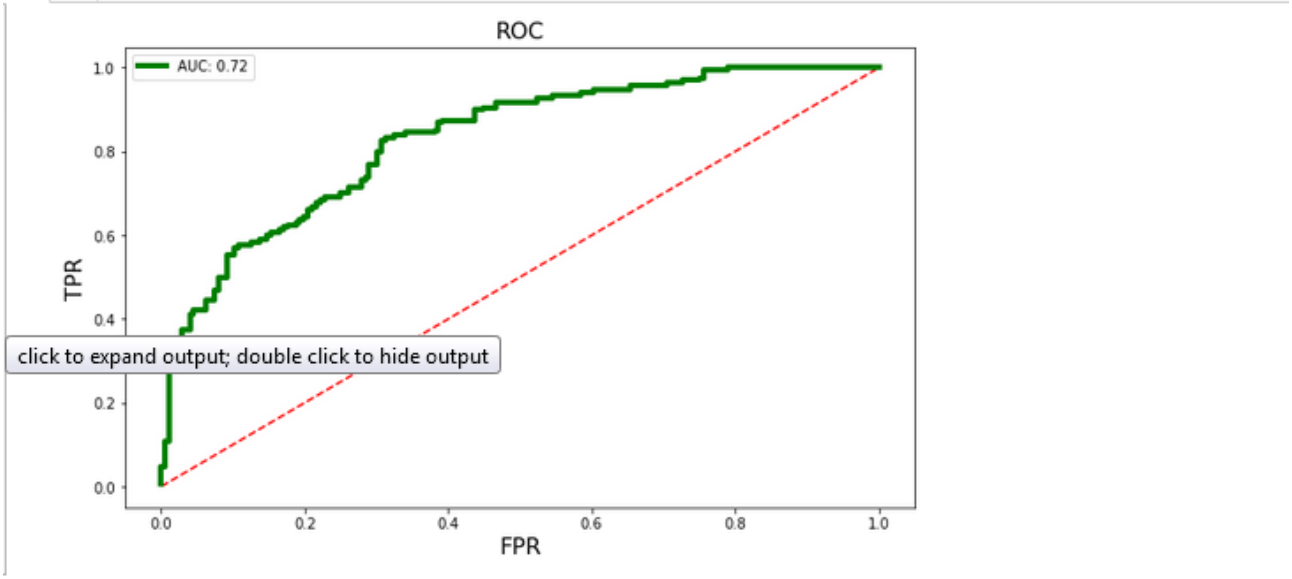


*Figura 26: Curva ROC*

A partir de ahí vamos a utilizar nuestro modelo, el clf, y vamos a decirle que haga una predicción con el conjunto de entrenamiento, pero no es una predicción normal, nos va a predecir la probabilidad de que sea 1 o 0. Como solo nos interesa la probabilidad de que sea 1 vamos a utilizar nada más esa parte de ahí. Vamos a decir: todas las probabilidades pero de que sea 1 solamente.

A partir de ahí Python nos facilita mucho el trabajo. Vamos a utilizar la función llamada roc_curve y le vamos a pasar el conjunto de datos "y" de prueba, y las probabilidades. Esto nos va a calcular automáticamente los parámetros necesarios, y a partir de nosotros vamos a utilizar esta segunda función que es el score. Una vez ya tenido eso lo que vamos a graficar como dijimos acá arriba es el TPR contra el FPR que afortunadamente ya nos cálculo Python en estas variables de aquí.

Entonces, los vamos a contrastar una contra la otra con una gráfica plot. Vamos a decir FPR contra TPR, vamos a decir que esa curva sea verde y que tengan como etiqueta esto que acabamos de crear aquí que es simplemente un mensaje que va a decir la precisión de nuestro modelo. Que sea un poquito más grueso ,que sea de tamaño 4 y a parte vamos a poner los ejes. Lo graficamos y nos da una curva similar a ésta.

Si utilizamos los mismos números aleatorios nos debe de producir la misma curva. Si cambian va a variar ligeramente. Ahora, ¿cómo se va a interpretar esta línea? Lo que pasa es de que si tuviéramos un modelo muy malo la línea verde estaría muy pegada a la línea punteada roja, sería algo muy

similar a esto. Por lo tanto, el área bajo la curva de ese modelo va a ser muy cercano a 0, es decir, entre más espacio o más área bajo la curva es mejor nuestro modelo como se dijo al principio.

Si tuviéramos algo muy similar a esto donde empieza en cero y sube directamente hasta uno podríamos decir que es un modelo perfecto. En real lo que puede estar pasando es que estamos sobre entrenando o que hay una variable multicolineal dentro de nuestro conjunto de datos con la variable de respuesta. Es decir, es una variable futura que no podría estar ahí. Entonces, esa variable tal vez está mal por agregarla.

La ventaja de este modelo de ROC es que nosotros podemos contrastar múltiples modelos al mismo tiempo. Por ejemplo, podemos tener nuestro modelo este, el verde, y podríamos estarlo contrastando contra otro, digamos que queremos cambiar nuestras variables estas. Tal vez no queremos duración ni edad, tal vez queremos edad y balance. Eso nos va a generar otro modelo que podemos graficar en la misma gráfica. Eso nos va a generar una línea que va a ir por arriba o por abajo de la línea verde.

Entonces, el modelo que nos va a servir más es el que va a ir por arriba de las curvas, es el que va a generar más área. Esto ya nos da algo visual del cual podemos decir: ah, un modelo es mejor que el modelo B. Esta precisión la podemos utilizar además para darnos una idea de qué tan bien le fue. Y hay una regla de dedo para decir "el modelo es bueno o el modelo es malo", y aquí está apuntado. Si esta precisión nos da entre 0 a 5, es un modelo extremadamente malo, es peor que utilizar un número aleatorio.

Si es alrededor de punto 0.5 es prácticamente el equivalente de lanzar monedas, es completamente aleatorio. Si es entre punto 0.5 y 0.6, pues es mejor que el aleatorio pero aún así es muy malo el modelo. 0.6 a 0.75 pues ya es algo regular. Lo esperado sería que fuera entre 0.75 y 0.9 que pues nuestro modelo, verdad, al menos el que tenemos ahora, pues no lo es. Entre 0.90 y 0.95 ya es un... es un modelo muy muy bueno. Y entre 0.95 y 1 es muy bueno para ser cierto, es donde deberemos tener cuidado con las variables que tenemos.

Esta es la curva roc, esto nos va a poder interpretar qué tan bien o qué tan mal se produjo nuestro modelo, qué tan buenas respuestas da.

Hay otra gráfica similar, que es la que veremos en la siguiente sección, que es la curva CAP. Esta nos va a ayudar a entenderlo un poquito en contexto de cuántas personas aceptaron y cuántas no a diferencia del ROC que solamente funciona con una variable entre 0 y 1.

## Curvas CAP

En esta sección vamos a analizar lo que es la gráfica CAP. La gráfica CAP, a diferencia de la ROC, nos va a permitir poder visualizar la calidad de nuestro modelo a través de números más reales. Lo podemos ver ya como personas que aceptaron nuestro préstamo a corto plazo o lo podemos ver como instancias que dijeron sí y no. Y no tanto en un porcentual, como la curva de ROC.

Ahora, si lo buscan en internet habrá mucha confusión a veces entre la curva ROC y la curva CAP. Hay que discriminar que no son lo mismo, hay una diferencia grande en las dos gráficas, pero vamos a ilustrarlo a través del ejemplo.

Vamos a iniciar. Esta gráfica CAP la vamos a ver con nuestro cuaderno en Python llamado "Mejorando los modelos". Arriba de eso está lo que es la curva ROC. Vamos a continuar, para

analizar esta gráfica vamos a simplificarnos un poco las cosas, vamos a hacer un muestreo de mil elementos para estar en números cerrados. Y vamos a utilizar dos variables la duración y la edad. Vamos a hacer el conteo de variables y vamos a ver que hay una proporción de 900 a 100, de 1000 personas 900 nos van a decir que "no" y 100 nos van a decir que "sí".

Ya es una proporción que nos da el conjunto de datos. Es decir que de cada 10, uno nos va a decir que sí. Ahora, eso lo podemos visualizar de alguna forma más intuitiva, vamos a generar ese mismo conjunto de datos y lo vamos a graficar en una figura. Vamos a hacer nuestra gráfica primero vacía, la graficamos y aquí está abajo. Entonces, nosotros ya sabemos que es la proporción de 1 a 9: por cada 10 personas que marquemos, una nos va a decir que si. Entonces, vamos a graficar esta gráfica. Vamos a cambiar de blanco a rojo y aquí la tenemos. El eje "x" nos va a decir cuántas personas hemos marcado, el eje "y" nos va a decir cuántas personas nos han dicho que sí.

Y como sabemos que la proporción es 1 sí, 9 no, pues cuando tengamos 0 llamadas pues vamos a tener 0 personas que nos han dicho que sí, evidentemente. Cuando tengamos 200, 20 nos habrán dicho que sí, cuando tengamos 1,000, 100 nos habrán dicho que sí. Es una relación constante que siempre va a estar pasando. Entonces, para poder contactar a nuestros 100 posibles clientes tendremos que haber marcado a 2,000 personas.

Ahora, nuestro modelo CAP, o nuestra curva CAP, lo que nos va a ayudar es poder distinguir qué tan rápido, a través de nuestro modelo, podemos llegar a estas 1,000 personas sin necesidad de hablarle a las 1,000. Ok, para ello vamos a utilizar este segmento de código de aquí. Para comentar y descomentar en mi caso en particular uso "ctrl + /" y esto nos lo descomenta, pero podríamos utilizar también el borrar comentario y a funcionar igual.

Ok, lo que va a hacer, vamos a considerar primero lo que sería un modelo perfecto. Vamos a graficarlo. Lo que haría esto es es decir que las primeras 100 personas que marcamos, las 100 nos dirían que sí. Eso sería el perfecto, poder identificar cuáles son las personas que nos van a decir sí. Entonces, para ello vamos a utilizar esta nueva función que no hemos utilizado hasta el momento, que suma acumulativa o "cumsum", y vamos a simplemente hacer el aditivo de 100 personas.

Esto lo vamos a agrandar a que sea un tamaño 1,000 para... la serie nos quedaría algo similar a 1, 2, 3, hasta llegar a 100, y a partir de 100 se va a quedar con ese valor constante porque ya contactamos a todas las personas. Esta variable la vamos a graficar y nos va a dar algo similar a esto.

Entonces, esta línea azul sería el equivalente a que pudiéramos predecir el futuro, sería una bola de cristal, estaríamos adivinos. Esto es muy difícil que se logre en la realidad, de hecho podríamos decir que es imposible. Si nosotros podemos crear un modelo que se asemeje a este en realidad habría algún problema con nuestros datos, que alguna variable sea posterior al proceso, que sea una variable del futuro la cual no podría ser, no podríamos generar un modelo de predicción estable. Entonces, tendríamos que eliminar esa variable. Ahora, vamos a descomentar la siguiente sección, y para eso vamos a aprobar nuestro modelo, qué tan bien se está comportando.

Si recuerdan nuestro modelo puede predecir las probabilidades. Vamos a decirle precisamente eso, que nos de predict_proba, o sea, que nos de las probabilidades del conjunto de datos reducido que tenemos aquí, cap_X, pero sólo queremos los casos "sí", vamos a abordar aquí en la variable "df_cap" en la columna "probs", de probabilidades, aquí es donde viene la diferencia con la curva ROC.

Lo que vamos a hacer es, como tenemos las probabilidades de que una persona nos siga sí o no, vamos a ordenarlos de cuál es la persona más probable que nos diga que sí y cuál es la menos probable que nos diga que sí, de esta forma nada más les vamos a marcar a las primeras "n" personas, digamos, a las primeras 30 personas que creemos que nos van a decir que sí, a las primeras 20, a las primeras 50, incluso los primeros 500, verdad. Entonces, hacemos precisamente eso, hacemos la predicción de probabilidades y luego ordenamos los valores de forma descendente.

Vamos a hacer la suma acumulativa de las personas que nos dijeron sí y lo vamos a contrastar contra las que nos dijeron que sí. A veces es difícil conceptualizarlo nada más viendo las líneas de código pero vamos a graficarlo.

```
df_cap = df.sample(1000,random_state=20)
cap_X = df_cap[['duration','age']]

plt.figure(figsize = (10, 6))
plt.plot([0,1000], [0,100], 'r--');
# ------------------------------------
perfect_model = np.cumsum(np.repeat(1,100))
perfect_model.resize(1000)
perfect_model[100:] = 100
perfect_model
plt.plot(perfect_model);
# ------------------------------------
df_cap['probs'] = clf.predict_proba(cap_X)[:, 1]
df_cap = df_cap.sort_values(by='probs',ascending=False)
probs = np.cumsum(df_cap.y).values
plt.plot(probs,c = 'g', label = label, linewidth = 4);
# ------------------------------------
plt.xlabel('Total de personas', fontsize = 16)
plt.ylabel('Total de inversiones (casos si)', fontsize = 16)
plt.title('CAP', fontsize = 16)
# ------------------------------------
plt.axvline(x=500, color='g', linestyle='--',alpha=0.5)
plt.axhline(y=probs[499], color='g', linestyle='--',alpha=0.5)

probs[499]
```

```
87
```

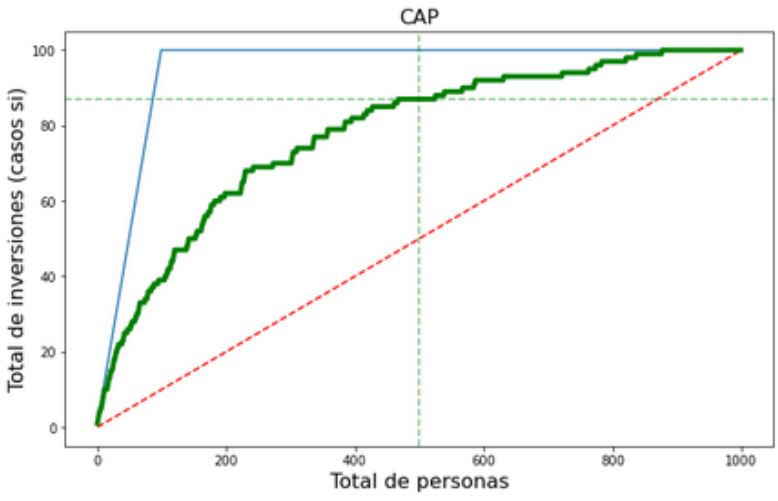


*Figura 27: Curva CAP*

Ok. Esta es la gráfica de nuestro modelo, la línea verde similar a la línea ROC pero es diferente en este caso. Lo que nos va a decir esta línea es de que cuando lleguemos alrededor de 400 personas si seguimos por aquí vamos a haber contactado a alrededor de 85 personas que nos dijeron realmente sí, es decir, lo que nos está mostrando esta curva es qué tan rápido vamos a llegar al valor en "y" que queramos, a las personas que nos van a decir que sí.

Un caso de dedo es tratar de decir: si contactamos la mitad de las personas, cuántas personas nos habrán dicho que sí. Entonces, vamos a descomentar esto que son los títulos de los ejes y vamos a probar a la mitad de las personas.

Ok, ya tenemos nuestros ejes y dijimos que vamos a probar a 500 personas porque tenemos 1,000, a la mitad, esto nos va a producir la intersección con la curva, y si lo seguimos nos va a dar alrededor de 85.

Lo que no quiere decir esta gráfica es que si contactamos a 500 personas, es decir, la mitad del total, vamos a haber contactado 85 personas, alrededor de 85 personas. El tomador de decisiones que esté a cargo podrá decir: no pues tal vez no me interesa marcar a las 1,000 porque la ganancia adicional de marcar esas 500 restantes es muy pequeña. Tal vez estamos gastando tiempo y recursos que podríamos aprovechar para algo más, o tal vez con haber contactado el 50% de los casos "sí" no es suficiente.

Entonces, con haber hecho 180 llamadas de las 1,000 pues es más que suficiente. Podemos descomentar esta última línea de aquí también y nos va a decir precisamente cuántas personas contactamos que nos dijeron que sí: 87.

Entonces hay alguna otra regla de dedo similar a la curva ROC para decir qué tan bien está nuestro modelo. De 0 a 0.6 es equivalente a aleatorio o peor que aleatorio, de 6 a 7 es un test malo, de 0.7 a 0.8 es un test bueno, decente, de punto 0.8 a 0.9 es un test muy bueno y de punto 0.9 a 1 es muy bueno para ser cierto, ahí es donde tenemos que preocuparnos de que no estemos agregando información que no debería estar ahí.

Entonces, esa es la diferencia principal entre la curva ROC y la curva CAP, donde la curva ROC en realidad nada más nos da valores numéricos de entre 0 y 1, y nos da probabilidades. En la curva CAP nosotros ya estamos trabajando con totales de personas. Por ejemplo, de aquí fueron 1,000 llamadas totales y siempre son los que nos dijeron que sí. Entonces, podemos ver ya claramente el valor del modelo ya en valores numéricos. Ya podemos asociar un valor económico a esas llamadas. Entonces, esa es la principal diferencia entre las dos curvas. En lo personal yo creo que la curva CAP determina mucho mejor la información pero hay quienes utilizan la curva ROC, y como científicos de datos deberíamos de poder ser capaces de interpretar ambas gráficas.

Ya utilizamos la curva CAP, pero con estas dos variables: "duración" y "edad". Hay algunas otras variables que no hemos utilizado que están acá arriba, que son, por ejemplo, no hemos utilizado el balance, o esta variable esta variable "pdays", o esta variable "previous". Entonces, hay diferentes variables que podemos tratar de utilizar en nuestro estudio. Ahora, como reto para ustedes les quiero decir que traten de cambiar las variables para ver cómo se comporta nuestra función CAP, o nuestra gráfica CAP, y ver si pueden generar un modelo que pueda predecir mejor o peor a estas características.

## Variables Dummy

En secciones anteriores ya vimos las curvas ROC y las curvas CAP que nos van ayudar a validar nuestros modelos de clasificación. Ahora, en los conjuntos de datos habrán notado que hay varias columnas de texto que no hemos utilizado hasta el momento. Ahora, estas las vamos a utilizar en esta sección y vamos a ver cómo se procesan para poder utilizarlas. Vamos a ver nuestro conjunto de datos.

Recuerden que estamos trabajando sobre el mismo cuaderno "mejorando los modelos", y es después de la curva CAP. Nuestro conjunto de datos tiene variables ya numéricas como edad, como balance o duración, pero tenemos algunas otras como, por ejemplo, el tipo de trabajo, o… su estado civil, su nivel de educación y estas variables no las hemos utilizado hasta el momento.

En general cuando estamos construyendo estos modelos predictivos el algoritmo asume que las variables son cuantitativas o son continuas. En este ejemplo, por ejemplo, la edad o el saldo y los datos pues en realidad vienen en forma categórica muchas veces, como les decía, el estado civil y el género entre otros. Ahora, es nuestra responsabilidad como científicos de datos transformar estos datos de una forma en que el algoritmo los pueda entender. Pues recuerden que los algoritmos en realidad, o los modelos proyectos, son funciones matemáticas.

Y hay dos alternativas para hacer esto: una es asignarles un valor secuencial, esto nada más es aplicable, por ejemplo, cuando tenemos meses del año, donde tenemos 1, 2, 3, 4, representando enero, febrero, marzo, abril, o cuando tengamos, por ejemplo, algo que tenga un orden natural como niveles de educación, podríamos asignar a primaria 1, a secundaria 2, a prepa 3, etcétera. Hay una secuencia natural en las cosas.

Muchas veces esto no es posible, por lo tanto hay que utilizar alguna otra técnica y de eso se trata esta sección. Vamos a utilizar una técnica llamada "variables ficticias" o "variables dummy" como se conocen en inglés, y esta es una variable artificial que representa una variable categórica de dos o más categorías.

Entonces, vamos a volver a echarle un vistazo a nuestro conjunto de datos. Tenemos trabajo, estado de civil, el nivel de educación. Pandas, internamente, estas columnas las conocen con algún tipo específico que la podemos visualizar a través de "dtypes".

Y aquí podemos ver, por ejemplo, edad es un entero, podemos ver que el balance es otro entero, el día es otro entero y notarán que todas las que tengan datos de texto se representan a través de este tipo de dato llamado objeto.

Bueno, esto lo podemos tomar a nuestro favor y poder filtrar a través de ese tipo de datos. Le podemos decir la Pandas que del conjunto original solamente tome las columnas donde el tipo de datos sea igual a objeto. Este objeto lo podemos accesar a través NumPy: NumPy, punto, objeto, y vamos a imprimirlas.

Entonces, en este conjunto de datos obtuvo que job, estado de civil y educación, entre otros, son tipos de dato objeto. Ahora, estos precisamente son las columnas de las cuales vamos a transformar. Esto, afortunadamente, Python lo hace de forma automática con una función llamada get_dummies. Entonces nosotros le podemos decir a Pandas: Pandas, por favor transforme estas columnas o...transfórmalas en artificiales del conjunto de datos df...todas las columnas que te indica aquí. Lo vamos a renombrar a df con dummies y vamos a imprimirlo. Ok. Verán que hay un cambio de columnas, ya no son tan poquitas como antes, de hecho ahora son 52 columnas cuando antes eran alrededor de, escribimos shape, de 17.

Entonces, ya hay un cambio grande ahí. ¿Qué es lo que pasó? Bueno, tomemos, por ejemplo, lo que es job... la columna job. Vamos a regresar esta celda como estaba antes, df.head( ), y job tenía estos valores: desempleado, de servicios, administración, o este último trabajo que es "obrero". Bueno, lo que hizo fue: tomó cada una de esas categorías, por ejemplo, administración, obrero, etcétera, las prepara de forma de que va a agregar ceros pero sólo va a agregar un 1 donde los usuarios sean

obreros, por ejemplo, en este caso, o sólo va a poner un 1 cuando si el es usuario de esa administración. Entonces, va a crear de forma artificial múltiples columnas, una por cada categoría. Entonces, lo podemos ver aquí, incluso si las imprimimos nos va a decir que hay una nueva columna para administración, para obrero, para emprendedor, para ama de casa, etcétera, una por cada una de las categorías, y eso no nada más funciona para job, sino dijimos que lo iba a hacer para todas las columnas.

Entonces, por ejemplo aquí tiene estado civil que es soltero, casado y divorciado, hay tres, y para el nivel de educación si es primaria, secundaria, terciaria o desconocida. Entonces, para cada uno de estos está agregando una columna nueva. Aquí hay que tener mucho cuidado, ¿por qué? Porque hay una especie de trampa, o problema al utilizar todas las columnas.

Y la trampa de las variables ficticias es un escenario en el cual dos o más variables independientes son multicolineal, es es decir, con una de las variables se puede predecir la otra. La consecuencia de utilizar estas variables es que nuestro modelo no puede utilizar correctamente esta información, si lo queremos ver de una manera simple es como que se contrarrestaran una con la otra estas variables. Entonces, no está produciendo problemas, sería como el equivalente a simplemente retirar la columna completa.

Afortunadamente hay una forma de arreglar esto, y esto es quitando una de las variables recién creadas, por ejemplo, en el estado civil dijimos que tenemos tres tipos: tenemos el divorciado, el casado o el soltero. Entonces, la forma de arreglarlo sería nada más agregando dos, por ejemplo, podemos quitar divorciado y nada más quedaría casado o soltero. Supongamos que hay un usuario que no tiene ninguno de estos dos elementos, que tiene un cero en la columna casado y cero en la columna soltero.

Entonces, podemos intuir que en realidad ese usuario debe estar divorciado porque la ausencia información en este caso también es información. Python nos puede hacer esta tarea de una forma automática, ¿por qué? Porque pues reconocen que existe ese problema y se regla simplemente agregando este atributo, "drop_first", que lo que va a hacer es, pues simplemente no agregar la primer columna.

Lo volvemos a repetir y listo, ya pasó a 43 columnas, y si volvemos a imprimir las columnas y vamos a nuestro ejemplo que era el estado civil va a estar casado y soltero, eliminó la divorciado pero sabemos que la ausencia de estas dos columnas va a ser equivalente a decir que es divorciado. Bueno, ahora ya tenemos nuestro conjunto de datos preparado, ya agregamos mucha nueva información que podemos utilizar para nuestros modelos predictivos. Entonces, vamos a probar nuestros modelos a ver si podemos construir algunos de mejor calidad.

Primero vamos a hacer un undersampling, ya lo vimos, entonces nada más lo vamos a ejecutar. La única diferencia es que al final le agregamos esta nueva...esta nueva instrucción, y le vamos a decir que todo el conjunto datos lo vamos a ver, no como números enteros, sino como flotantes, esto va a acelerar internamente los cálculos de Python.

Ahora sí. Y ahora vamos a dejarles un ejercicio para que puedan probar qué tan efectivo es utilizar esta nueva información. Lo que vamos a hacer es, ya tenemos nuestro script que tiene todas las instrucciones para generar modelos y para graficarlos a través de curvas ROC. Ok, lo que vamos a hacer es tratar de describir cada uno de estos comentarios, qué es lo que hace el segmento de código siguiente y, una vez que hayamos hecho eso hayamos comprendido un poco qué es lo que está haciendo el script, vamos a tratar de mejorar nuestro modelo cambiando las columnas que

utilizamos por ejemplo podríamos decir que en vez de duración vamos a utilizar la variable mes de septiembre porque ahí es una variable válida y con ello vamos a ver si podemos construir un modelo más efectivo que los que hemos construido anteriormente les dejamos esta duración y edad porque son los mismos variables que utilizamos anteriormente para poder controlar estar que también aunque tan mal se comporta el nuevo algoritmo.

## Selección de características relevantes

En la sección anterior estuvimos trabajando sobre la variable ficticia. Ya agregamos la capacidad de las variables categóricas a nuestros modelos de regresión. Antes de eso vimos las curvas ROC y las curvas CAP para poder validar nuestros modelos. Seguramente en el ejercicio anterior, cuando estuviste van a buscar variables, estuviste haciendo iteraciones, agregando variables y quitando variables, viendo si esto subía o bajaba el error. Eso es un proceso que puede ser tardado como te habrás dado cuenta.

Para ello, para automatizar un poco esto, hay funciones en Python que nos van a permitir hacer esta selección de quitar y poner variables para encontrar el mejor modelo posible. Esto nos va a servir para discriminar muchas variables de una manera rápida. Vamos a seguir trabajando sobre nuestro cuaderno de Python llamado "mejorando los modelos", y vamos a ir directamente a la sección que dice "selección de características".

Vamos a hacer un poco de lectura. Una vez que ya trabajamos con las variables categóricas nos queda un conjunto con tres variables. Seguramente muchas de las que generamos son irrelevantes, no sirven para poder hacer predicciones. Estas variables que no funcionan nos pueden dar problemas a la hora de crear nuestros modelos. Las variables irrelevantes pueden producir diferentes cosas, o diferentes efectos.

El primero de ellos es que desperdician RAM, memoria RAM, que es la memoria física de la computadora. A escalas pequeñas tal vez no nos preocupe tanto, tal vez no afecte tanto, pero cuando escalamos ya a niveles reales donde hay miles o millones de valores, pues cualquier columna que podamos quitar nos va a beneficiar a nuestra memoria RAM, es decir, libera espacio.

Otro problema de las variables irrelevantes es que actúan como ruido. "Ruido" se refiere a esas alteraciones naturales que existen en los conjuntos de datos, por ejemplo, los errores de dedo al estar tipeando los datos o que hay una lectura equivocada, todo esto funciona como ruido y las variables relevantes también incluyen ese ruido natural. Si las incluimos a nuestro modelo y no nos benefician en nada, pues nos están afectando nuestro poder predictivo. Además estas variables, si las estamos agregando y estamos entrenando con ellas cuando escalamos a millones al igual que en el primer punto, nos van a incrementar el tiempo de entrenamiento, nos va a afectar ya en ese tipo de características.

Entonces, la solución a esto es eliminar esas variables irrelevantes. Entonces, la solución al problema que estamos mencionando es poder seleccionar correctamente las variables relevantes. Es básicamente el proceso de seleccionar variables o características más relevantes de un conjunto de datos.

Hay básicamente tres formas de hacer esto: la primera es a través de métodos de filtrado, son aquellos que utilizan medidas estadísticas para poder determinar qué variables son buenas o qué variables son malas, de esta forma podemos determinar... nada más tomar las variables que son

buenas y utilizarlas para nuestro modelo. En la siguiente sección vamos a ver un ejemplo de un método de filtrado. También están los métodos de envoltura o de wrappping que se llaman en inglés. Son aquellos modelos que utilizan modelos predictivos para determinar la calidad de las variables, no a través de una medida estadística sino a través de modelos predictivos. Esto lo que vamos a ver en esta sección. El que no vamos a ver, al menos en esta serie de secciones, son los métodos integrados, son aquellos incorporan... se incorporan al proceso de selección o de entrenamiento, en el modelo predictivo. Y no lo vamos a ver porque en realidad ya lo vimos, aunque no le hayamos puesto atención o no lo hayamos enfocado a este aspecto. ¿Por qué? Porque lo vimos a través de los árboles de decisión.

Los árboles de decisión iban tomando decisiones binarias "sí" o "no", dependiendo de una variable. Las variables que fueron seleccionadas en el proceso de construir el árbol fueron seleccionadas a través de un método integrado que es el árbol. Entonces, eso ya... no hay mucho que hacer, ya está construido dentro del algoritmo. Entonces, por lo tanto, no vamos a darle mucho énfasis. Entonces, les mencionaba que en este video vamos a ver los métodos de envoltura que generalmente se realizan de una manera similar a prueba y error, y de una manera iterativa, estamos probando constantemente con modelos nuevos si funcionan o no funciona.

Y esto lo hace incorporando o quitando variables en cada uno de los ciclos de prueba y error. Esto se conoce como una selección paso a paso o stepways en inglés. Generalmente hay una confusión a la hora de manejar estos métodos y se confunden mucho con un problema de reducción de dimensionalidad.

Aunque ambas técnicas buscan reducir el número de características de forma de que utilizamos menos variables, lo que hace la reducción de la dimensionalidad es que transforma el conjunto completo de datos. En este caso, en el de nosotros, de selección de métodos por envoltura, lo que hacemos es eliminar información de nuestro conjunto de datos y de esa forma se reduce. La reducción dimensional tiene su aplicación pero no la vamos a tomar en este momento. Para poder hacer nuestros métodos de envoltura existen dos formas principales.

En realidad existe una tercera pero vamos a dejarla por el momento. La primera de ellas es el... selección hacia adelante o forwards selection, y vamos a verlo a través de este diagrama. Primero iniciamos con una variable, seleccionamos una variable al azar y construimos un modelo predictivo y luego lo evaluamos.

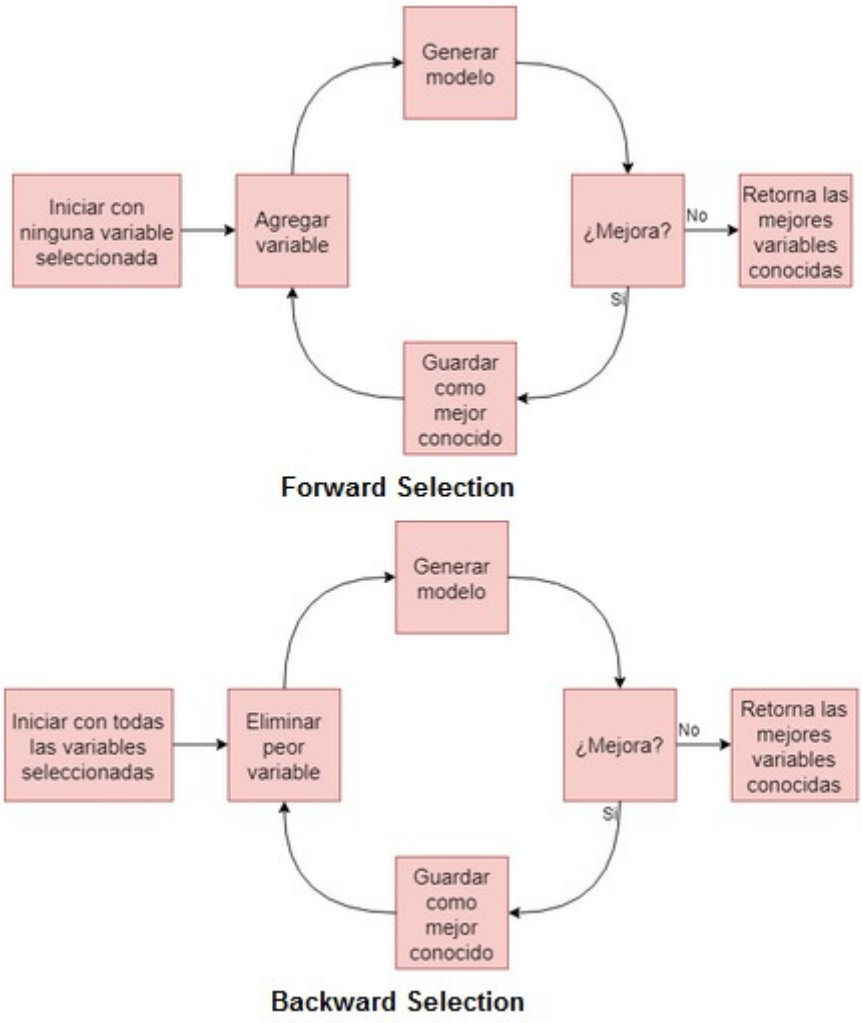


*Figura 28: Forward y Backward Selection*

Ok, una vez que ya tenemos un modelo de una variable vamos a agregar una segunda variable. Generamos el modelo y preguntamos: ¿este nuevo modelo es mejor al anterior? Si sí es mejor vamos a considerar que esa variable es relevante porque aportó al problema. Entonces, lo vamos a guardar como el nuevo mejor conocido.

Vamos a ir y agregar otra variable, vamos generar un nuevo modelo y si ese modelo mejor al que acabamos de crear, al que era el conocido como el mejor, vamos a reemplazarlo, efectivamente agregando una nueva variable. Este proceso va a ir repitiendo hasta que el nuevo modelo ya no mejora, es decir, ya encontramos las mejores variables conocidas, al menos bajo esta técnica.

El backward selection es algo muy parecido pero al revés: parte de un modelo que tiene todas las variables y va eliminando variable por variable hasta que deje de mejorar. Ambos tienen su aplicabilidad y ambos son muy importantes.

Hay una tercer técnica que es cuando hacemos forward selection, o empezamos a agregar variables hacia adelante, y luego si ya no mejor empezamos a quitar variables. Todos estos métodos son muy importantes y se utilizan en diferentes software. Afortunadamente estos métodos no los tenemos que programar, ya están dentro del lenguaje de programación Python, específicamente sobre la librería sk-learn, y en específico aunque no maneja estos dos métodos que acabamos de ver, maneja uno

llamado recursive feature elimination. Lo que hace es algo muy parecido al backward selection pero lo hace de una manera en la cual puede agregar de dos variables, de tres variables, cuatro variables, etcétera.

Para ello utiliza un modelo predictivo que el que vamos a estar utilizando la regresión logística. Este es un método que es un poco más tardado, ¿por qué? Porque en cada iteración está construyendo un nuevo modelo, lo está entrenando y lo está probando. Entonces, es algo que va a estar haciendo constantemente, y si tenemos cincuenta y tantas variables lo va a hacer cincuenta y tantas veces. Entonces es algo con lo que hay que tener cuidado.

Para acelerar un poco el entrenamiento lo que vamos a hacer es, vamos a escalar nuestros datos. Los vamos a escalar a que estén en un rango entre 0 y 1, esto internamente acelera el procesamiento, y esto lo vamos hacer a través del MinMaxScaler, lo que hace nada más: toma el mínimo y el máximo, y los ajusta a un rango de 0 y 1. Es muy sencillo de utilizar.

Por eso preparamos nuestros datos en la variables "x" y "y" . Además crearemos un método que escalen las variables que sea de la de la librería MinMaxScaler. Y vamos a usar uno específico para "x" y uno para "y". Esto no es realmente necesario pero personalmente encontrado que es más fácil manejar dos diferentes, sobre todo cuando queremos retornar los valores a su escala real.

Ok, así creamos nuestro método que va a estar haciendo el escalado vamos a entrenar y vamos a transformar los datos. Vamos a entrenar con la... y transformar el conjunto de datos "x", y lo vamos a guardar como nueva "x" y lo mismo lo vamos a hacer con "y". Vamos a crear una función que escale para "y", y vamos a escalar los datos de "y". Aquí es necesario que transformemos el conjunto de datos con un reshape.

Ok. Después de eso hacemos un spirit como siempre lo hemos hecho. Ahora sí, ya tenemos nuestros datos preparados, y vamos a utilizar el RecursiveFeatureElimination (RFE) o eliminación de características recursivo. Esta técnica particularmente utiliza un modelo de estimación que vamos a llamarle "estimator". Si se fijan no lo estamos entrenando, simplemente lo creamos, e internamente nos lo entrenara múltiples veces. Ahora sí utilizamos la función RFE, le pasamos el estimador que es regresión logística, le vamos a pasar este 8. El 8 lo que quiere decir es el número de variables finales que queremos. Esto es un parámetro ajustable y tal vez queremos dos variables, tales queremos cinco variables, eso es algo que ya les encargaremos a ustedes que lo preparen o que lo ejerciten.

Además podemos decirle cuántas variables queremos que elimine en cada iteración del algoritmo, en particular vamos a decirle que queremos de una en una, pero podemos acelerarlo poniéndolo de dos en dos, de tres en tres, etc. Ahora, a esta selección de variables hay que entrenarla y lo vamos a hacer con el conjunto de entrenamiento "x" y "y", y vamos a imprimir dos atributos de este selector, vamos a imprimir un vector que se llama de "soporte" y uno de "ranking". Vamos a ejecutarlo a ver qué es lo que hace. Si contamos el número de verdaderos verán que es uno, dos, por aquí tenemos el tercero, cuarto, quinto, sexto, séptimo y octavo.

Ese 8 coincide con el parámetro que le pasamos. Entonces, lo que está haciendo es... nos está diciendo cuáles variables son las importantes o las relevantes.

El orden es el mismo del conjunto datos original. Vamos a imprimirlo aquí arriba para poderlo visualizar. (df.head( ) )

Aquí está, ahora nos está diciendo cuáles son las variables que son relevantes para el problema. Nos está diciendo que la cuarta variable relevante, también la que está aquí hasta acá abajo, todos los verdaderos son los relevantes y respetan el mismo orden nuestro conjunto de datos original. Además nos da el ranking, el ranking nos va a decir en un 1 los que son de mayor prioridad, 2 va a representar los que son de menor prioridad, 3, 4, así hasta el número de variables. Ok. Pero como las podemos visualizar estas variables, cuáles son. Bueno, podemos decirle que nos dé ese vector de soporte y vamos a guardarlo como "mejores variables" o "best variables", y vamos a decirle que al conjunto de datos reducidos nos localice todas las filas pero nada más las columnas que se encuentran dentro de esa variable de mejores variables, y la vamos a imprimir... la cabecera. Ok.

Entonces, ya nos dio un conjunto de datos reducido. Nos está diciendo que "duración" es importante, la "campaña" es importante, si tenemos un contacto telefónico, si estamos marcando en el mes de enero, en el marzo, en el de mayo, el de noviembre. También parece que son variables relevantes, y esta última variable que es "poutcome". En específico si tiene el valor de "other".

Ok. Ahora vamos a probarlo a ver cómo nos da nuestro modelo. Vamos a construir una regresión logística con este conjunto de datos reducido, vamos a pasarle las variables "y", y ya vamos a crear la predicción. Ya lo guardamos como y_test y ahora vamos a graficarlo en una curva ROC.

Lo graficamos, y nos dio un valor de 0.82. Si recordamos en nuestros ejercicios anteriores, vamos a nuestra curva ROC, teníamos un valor de 0.72. Entonces, al haber agregado variables dummy y haber hecho una selección de características subimos de 0.72 a punto 0.82, ganamos 10 puntos porcentuales que es demasiado en este tipo de problemas, ya estamos en un muy buen rango y tenemos un modelo predictivo efectivo.

Este conjunto de herramientas nos permiten poder mejorar modelos que se comportaban de una manera regular a una manera ya eficiente la cual podemos utilizar para poder hacer predicciones a futuro. En ésta sección vimos lo que es un método de envoltura, en la siguiente sección vamos a ver un ejemplo de un método de filtrado que corresponde a un problema de regresión.

## Matriz de correlación

En la sección anterior hablamos de los métodos de envoltura para selección de variables, se mencionó que también existen los métodos de filtrado. Ahora, vamos a ver uno de estos en especial para un problema de regresión.

Entonces, vamos a abrir el cuaderno "matriz de correlación" que se debe encontrar dentro de sus archivos. Entonces, en la sección anterior dijimos que los métodos de filtrado son aquellos que utilizan medidas estadísticas para ordenar las variables por su significancia. En específico uno de ellos es la correlación y vamos a hacerla a través de la matriz de correlación.

Primero vamos a definir la correlación. ¿Qué es? Se refiere a cualquier asociación estadística, pero más comúnmente se refiere al grado en el cual dos variables se encuentran relacionadas linealmente. Estos valores generalmente oscilan entre -1 y 1, y hay dos componentes principales al hablar de una correlación.

Primero, es la magnitud, qué tan alto o qué tan bajo es esta correlación, y para nuestro caso es mucho más significativo que se encuentre dentro... cerca de -1 o 1, que cerca del 0.

Además el signo es importante. Si el signo es positivo hay una relación directa entre el par de variables. Ahora, para este problema vamos a utilizar un conjunto de datos que hayamos utilizado, que es la predicción de las casas. En particular en este conjunto de datos no tenemos que descargar nada, es información que ya está cargada dentro del librería sk-learn. Vamos a utilizar de la librería "data sets" dentro de sk-learn, el conjunto de datos load boston, que es... carga el conjunto de datos de casas, la información de casas de la ciudad de Boston, casas que se encontraban en venta específicamente. La cargamos a través de esta instrucción, load_boston, y lo vamos a crear en un conjunto de datos de Pandas. Le vamos a decir: boston.data, como información, las columnas van a ser boston, pero con el nombre de las características, y vamos a agregarle una última columna que es el valor de las casas, que se establece a través de la variable MEDV, que es valor medio, una vez hecho esto vamos a hacer la matriz de correlación, que esta nos la calcula automáticamente Pandas.

De hecho si recuerdan en secciones anteriores ya hemos utilizado la correlación, pero no hemos enfatizado en ella. Ahora sí, ya tenemos nuestra matriz. Entonces, dijimos en nuestra definición que queríamos buscar los valores que sean muy negativos o muy positivos. Ahora, inmediatamente salta a la vista que hay una diagonal con 1's. Esta en realidad no nos sirve, ¿por qué? Porque ésta mide la relación entre la misma variable, es decir, por ejemplo, esta variable CRIM contra esta variable CRIM. Evidentemente pues van a ser totalmente correlacionadas porque es la misma variable.

El objetivo es buscar variables diferentes y su relación. Entonces esta diagonal principal la vamos a ignorar. Y si se fijan la matriz en realidad es un reflejo, el triángulo superior es equivalente al triángulo inferior, entonces no tenemos que estar buscando en todo, nada más con un triángulo es más que suficiente. Ahora, estar buscando los valores o muy negativos o muy positivos puede ser difícil cuando tenemos mucha información en una tabla. Se puede dificultar, podemos buscar, por ejemplo, aquí hay un 0.73, pero se pierde, por ello Python nos facilita una forma de poderlos graficar fácilmente y esto es a través de un mapa de calor.

```
import seaborn as sns
import matplotlib.pylab as plt
%matplotlib inline
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");
```

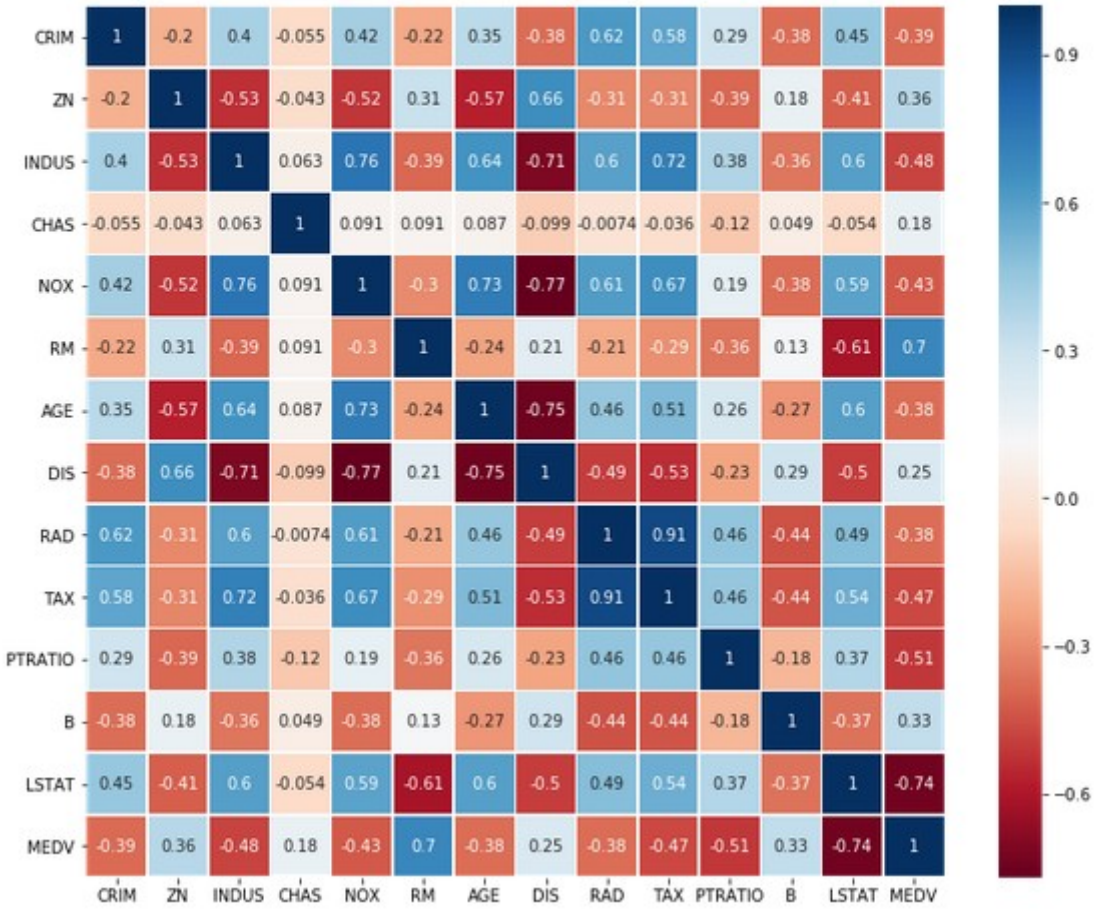


*Figura 29: Mapa de calor de correlaciones*

Aquí está un ejemplo vamos a volverlo a graficar, pero es a través de la librería seaborn que nos da esta función, y matplotlib para poder trabajar con la gráfica directamente. Vamos a hacerla que sea un poquito más grande diciéndole que el tamaño de la figura sea de 10 x 12. Estas generalmente son pulgadas pero se puede cambiar internamente. Vamos a decir que haga un mapa de calor, ¿con qué? Con la correlación, la que calculamos en un paso anterior, aquí la nombramos, corr, vamos a darle un ancho de líneas para que se marquen estas líneas blancas, vamos a hacer que tengan notaciones, es decir, que tengan estos números y vamos a decirle que el mapa de color es este conjunto de caracteres de aquí.

En internet podrán buscar algunos que otros conjuntos de colores o mapas de colores, pero este creo que ilustra muy bien la diferencia entre muy positivo y muy negativo, y valores neutrales que son casi cercanos a blanco.

Ahora salta a la vista estos cuadros muy rojos y muy azules. Por ejemplo, tenemos que este variable DIS tiene los valores de -0.75 o esta TAX tiene valores de 0.91. Ahí estamos midiendo la relación que existe entre pares de variables. Pero recordemos... demos un paso atrás y recordemos cuál es nuestro problema de predicción: queremos, con las variables comunes, o independientes, predecir una variable dependiente.

Nuestra variable dependiente era el costo de las casas. Entonces, en realidad lo único que nos interesa es esta columna de aquí. Entonces, vamos a ponerle especial atención a ella. De aquí salta a la vista dos variables: una, que es este azul que marca 0.7, y este otro de acá de -0.74. Estos valores corresponden a RM y LSTAT, vamos a graficarlas. Ok, vamos a hacer esta gráfica de dispersión, en el eje de las "x" vamos a poner el número de cuartos y en el eje de las "y" el precio de las casas. Entonces, lo que nos representa esta gráfica es que a medida que tenemos más cuartos generalmente crece el costo medio de la casa. Entonces, es una relación que es positiva, tal y como lo indica nuestra matriz de correlación que aquí está. Aunque no es perfecta es un buen valor para a partir de ahí.

Nuestra otra variable era LSTAT que es un nivel de pobreza promedio de la zona. Entonces, lo que nos dice es que a medida que crece la pobreza alrededor de la propiedad el valor de la casa decrece. Es una correlación negativa pero que nos puede ayudar para predecir el valor de la casa.

Bueno, probemos algún otro valor. Por ejemplo, tomemos el DIS, que el DIS nos marca una correlación de 0.25, no es muy alta, si la graficamos efectivamente podemos ver que no hay una correlación clara, tal vez al principio de los datos podemos ver que a medida que crece de 2 a 4 pues empieza a crecer el precio de la casa. Pero ahí es donde se rompe la relación.

Ya todos los datos están dispersos y no hay una relación clara. Entonces, esto sería una columna que no tiene una correlación con nuestra variable de predicción. Una vez que ya hemos hecho esto y vemos que podemos seleccionar algunas variables a partir de su correlación vamos a simplificar un poco nuestra gráfica de calor para que esta visualización sea mucho más efectiva.

Dijimos que sólo nos interesa el valor de las casas como variables de respuesta. Entonces tenemos... vamos a crear la figura de 5 a 10, una relación de 5 a 10, y vamos a agregarle un data frame que va a contener solamente la correlación en específico de la columna "precio de las casas". Vamos a agrega anotación, vamos a agregar las mismas líneas y el mismo color, y lo que nos produce es este mapa de calor reducido donde nada más incluye las variables o las correlaciones que nos interesan y salta a la vista cuáles sí funcionan y cuáles no funcionan.

```
plt.figure(figsize=(5,10))
sns.heatmap(pd.DataFrame(corr.MEDV), annot=True,linewidth=0.5,cmap="RdBu");
```

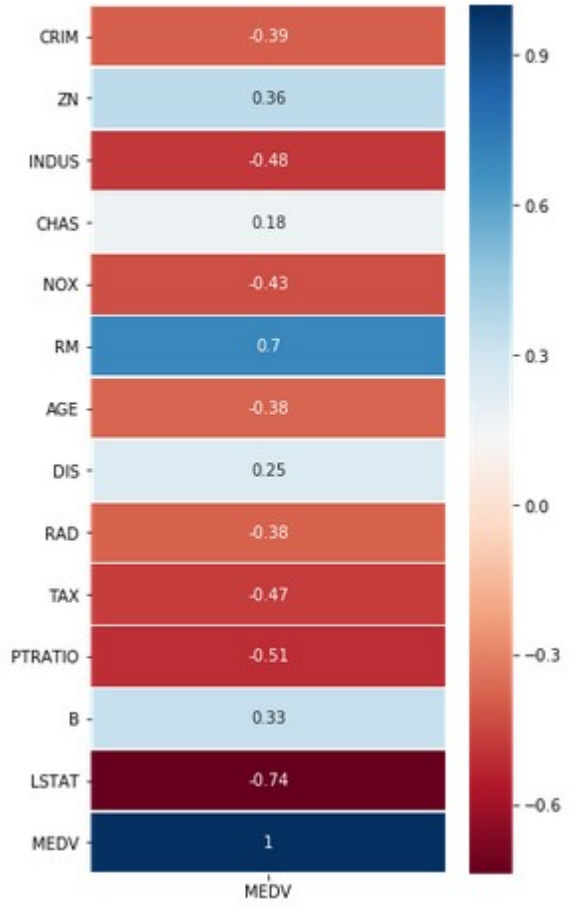


*Figura 30: Mapa de calor reducido*

Entonces, si quisiéramos poder seleccionar las variables podríamos, por ejemplo, decir que 0.7 es importante, LSTAT es importante, y de ahí digamos que queremos seleccionar tres variables, y la siguiente que escogería sería esta, PTRATIO porque es la que sigue en nivel de significancia con un valor de -0.51, y a partir de eso yo podría hacer la selección de las tres variables y efectivamente estaría haciendo una selección por método de filtrado, lo filtre a partir de correlación.

Entonces, esa es otra técnica con la cual podemos seleccionar variables. Con esto terminamos lo que es la semana, estuvimos viendo diferentes técnicas para poder mejorar nuestros modelos.

## Teoría sobre Métodos de Regresión y Ensamble

En esta sección vamos ahora a llevar a la práctica todos los métodos de regresión que hemos visto para un problema ya más real, un problema nativo de una problemática que actualmente se está viviendo en una minería. Hasta ahora todos nuestros datos han sido datos seleccionados y manipulados de tal manera que los modelos que estuvimos trabajando siempre daban una muy buena representación o aproximación.

Desafortunadamente en la vida real los datos no suelen tender en una forma específica o en una forma gráfica gráfica y fácil de ver. En esta sección y en la que sigue vamos a manipular un conjunto de datos un poco más complejos, un poco más grande y difícil de manipular.

Para esto hemos preparado una libreta, vamos a abrirla. La libreta se llama métodos de regresión y ensamble. Vamos a iniciar viendo qué es el problema que vamos a abordar en esta libreta. Vamos a leerlo juntos.

Dice: los datos provienen de una planta minera, el objetivo principal de estos datos es poder predecir la impureza en el concentrado del mineral extraído. Esta impureza es medida a cada hora. Con las predicciones pudiéramos ayudar a los ingenieros con una alerta temprana para que puedan tomar acciones correctivas. ¿Qué datos nos interesan? Nos interesa el porcentaje del concentrado de óxido de silicio, el concentrado de acero, el concentrado de sílice es la impureza del mineral del hierro que debe eliminarse y el proceso actual de detección de sílice lleva muchas horas.

Con la ayuda de algunos análisis y modelados de datos podemos dar una buena aproximación del concentrado de sílice que reduciría mucho tiempo y esfuerzos necesarios para poder procesar el mineral de hierro.

Nosotros estamos acostumbrados a empezar en la idea de que un robot simplemente es una máquina y es un objeto físico que interacciona con el medio pero también los robots se pueden identificar con unos agentes de software muy parecidos que se llaman bots. En este tipo de sucesos ellos toman información del medio, empiezan a procesar información, empiezan a modelarla, y con eso empiezan a trabajar y a subprocesar cosas que al ser humano le serían muy difícil de realizar la idea detrás de esta libreta es que veamos cómo un bot podría ayudar, a partir de recolectar datos, a realizar el proceso previo para que los ingenieros puedan tomar decisiones y poder hacer acciones correctivas ante un proceso actual.

Pero bueno, basta de hablar tanto de la descripción del problema y vamos a centrarnos ahora sí en la libreta. ¿Qué vamos a empezar haciendo? Bueno, esta libreta ya es un poco más aproximada a una forma de programación más ordenada y limpia donde vamos a tratar de agarrar muchos conjuntos de softwares, y aprovechando que ya tenemos una noción de qué hace cada uno de los modelos, lo vamos a buscar empaquetar en funciones un poquito más sencillas para no estar reutilizando tantas líneas y reescribiendo y reescribiendo.

Ahora que lleguemos a esta sección del código lo van a ir viendo. La primera parte de nuestra libreta es importar todas las librerías que vamos a estar ocupando a lo largo de esa libreta. Las primeras librerías que vamos a importar son NumPy y Pandas para poder manejar el conjunto de datos. Después matplotlib y seaborn. Estas las vamos a ocupar para poder realizar todas las parte gráfica de las de... graficar los modelos, graficar el conjunto de entrenamiento, graficar el conjunto de prueba. Después vamos a ocupar a sk-learn para importar las métricas y la selección de train_test_split y el pre-procesamiento.

Pero bueno, vamos a empezar. Entonces, importamos todas las librerías que necesitamos. Vamos a hacer la primera parte. En nuestra primera instrucción de lectura dice: el Data Frame va a ser igual a lo que leas del archivo, y ahí está la ruta, y al final hay un .drop_duplicates. ".drop_duplicates" nos va a permitir tirar todos los datos que sean renglones completamente iguales.

Una vez que ya procesamos esto vamos a eliminar la columna que dice date y la columna que dice "porcentaje de acero... de concentración de acero". Ahora sí, ya de ese Data Frame yo puedo

agregar, o toda la información, o una muestra. Por ejemplo, aquí, donde dice data frame es igual a .sample de 10,000 ( df = df.sample (10000) ) vamos a agarrar 10,000 de los campos que hay ahí y nos vamos a quedar solamente con esos. Podríamos trabajarlo con todo pero, para hacer un ejercicio un poco más fluido, con 10,000. Muestro los cinco primeros para que nos hagamos una idea de qué contiene este conjunto de datos y tenemos que al menos por cada uno de los campos hay 22 columnas que nos están dando de información. La opción "describe" nos puede dar cuántos datos hay, cuál es la mediana, cuál es la desviación estándar, cuál es el mínimo, el primer cuartil, segundo cuartil, tercer cuartil, el valor máximo, son cosas que a lo mejor ya hemos ocupado en otras secciones pero simplemente nos permite tener una noción un poco más fluida de qué datos está representando ese conjunto.

Pero bueno, ahora que ya lo tenemos en memoria ya podemos empezar a trabajar. Vamos a empezar a trabajar con estos datos, y para eso vamos a prepararlos en dos grupos importantes: las "x" ,que van a ser las variables de entrada, que van a determinar la "y", que es lo que queremos predecir. Bueno, ¿cómo lo vamos a preparar? Lo vamos a separar en dos: en "y" vamos a grabar el porcentaje de silicio, y en "x" vamos a grabar todo lo que no incluya a ese porcentaje. Entonces, ya tenemos aquí a la "x" y a la "y", y para poderlo estar trabajando de una forma sencilla y práctica vamos a escalar esos datos. Vamos a escalar esos datos ocupando la función de preprocesamiento que viene sk-learn, que es MinMaxScaler, y vamos a grabar la "x" escalada en una variable que se llama así scaler_X, y la "y" escalada en scaler_y.

Ahora, al reproducir esta parte ya podemos definir que mi "x" va a ser el conjunto de haber escalado esto con la transformación adecuada, y la "y" va a ser la transformación que ocupó para poder redistribuir los datos entre -1 y 1. Ocupando el sk-learn tengo la opción de poder decir cuál es el conjunto X_train, y_train, X_test, y_test de una forma sencilla a través de la función train_test_split.

En la función train_test_split recibe de parámetros cuatro cosas: la X, que se conjunto de los datos de entrada, la "y" que son las salidas, el porcentaje de la prueba y la semilla aleatoria. Yo sé que cambiando la semilla podríamos tener diferentes resultados. Ahorita déjenla fija en 103 para que sus resultados sean equivalentes a los nuestros, pero esa es una de las primeras cosas con las que podrían estar moviéndole, cambiándole la semilla para obtener otros datos. Pero bueno, ahora vamos a mantenernos con éstos.

Nuestra siguiente sección es definir una función que vamos a poder estar reutilizando a lo largo de esta libreta que se llama "plot" Plot es la palabra en inglés que nos representa "graficar". Bueno, esta función va a recibir cuatro parámetros de entrada: la "y" de entrenamiento, la "y" de predicción de entrenamiento, la "y" de prueba y la de prueba de predicción. Pero bueno, vamos a ver qué hace con estos cuatro parámetros.

La primera parte es para definir el tamaño y los límites que va a tener nuestra gráfica. Nuestra gráfica va a ser una figura de tamaño 10 x 10, un cuadrado. Después, los ejes van a ser, en el eje horizontal desde 0 hasta 1 y en el eje vertical desde 0 hasta 1. Y en la gráfica va a tener una línea que va a atravesar exactamente la diagonal punteada roja para una referencia.

Estas tres instrucciones de la función siempre van a realizar lo mismo. Van a realizar lo mismo porque no están atados a ninguno de los cuatro parámetros que vamos a recibir. Pero ahora sí, ¿qué va a hacer con lo que recibió?. Con lo que recibió va a generar dos scatter y en cada uno de esos scatters, o gráficas de dispersión, va a comparar la parte real contra la parte de la predicción.

El primer scatter va a comparar la "y" de entrenamiento real contra la "y" de entrenamiento de vuelta por la predicción. El segundo scatter va a comparar la "y" de prueba real contra la "y" de prueba de vuelta por la predicción.

Entonces, en ambos casos vamos a mostrar dos scatters a partir del conjunto de datos. Después, adentro de la misma función sigue habiendo instrucciones. Lo que siguen dice: preparamos las etiquetas. En la parte donde dice plot punto legend ( plt.legend) nos... va a mostrarle todas las etiquetas en la gráfica donde xlabel, el eje de las "x" va a ser la "y" real, ylabel va a ser la "y" de predicción.

Ahora, ya que tenemos eso lo último que falta es, aprovechando que ya tenemos una función custom completamente a la medida, es que una vez que salga la gráfica nos reporte también el error para que no tengamos que estar haciendo una y otra vez esas instrucciones, sino que se ejecuten una vez y ya nos devuelva todo eso.

El error lo calculamos a partir de el error cuadrático medio con la función mean_squared_error. Le pasamos los parámetros que están en pantalla e imprimimos: el error cuadrática medio es este error.

Ya que tenemos esta función, es importante que la reproduzcan para que esté en memoria, la podemos estar llamando una y otra y otra y otra vez.

El primer modelo que vamos a programar que va a enfrentar este conjunto de datos es un modelo de regresión lineal. Desde ese caldero vamos a importar el modelo de regresión lineal y en la regresión va a ser lo que devuelve a la función LinearRegression al momento de ponerle el .fit con los parámetros de "x" de entrenamiento y "y" de entrenamiento, y después, con esa regresión que generemos, vamos a intentar predecir ocupando las "x" de entrenamiento lo que serían las "y" de entrenamiento predichas y también vamos a intentar predecir los valores de validación, ¿cómo? Bueno, a través de los valores de la "x" de prueba van a generar al pasar a través través de nuestros modelos de regresión las "y" de prueba de predicción.

Entonces, vamos a reproducirlo. En esta línea ya se hizo nuestro modelo, ya empezó a hacer la regresión lineal, empezó a intentar predecir valores tanto de entrenamiento como de los... con los valores de test y vamos a graficarlo. Graficamos de una forma sencilla, simplemente llamando a la función plot que ya habíamos definido previamente, y la función plot recibió los cuatro parámetros y nos devolvió que el error cuadrático medio es de 1.050. Aquí tenemos nuestros datos. Y quizás nuestro modelo de regresión no se parece en lo más mínimo al modelo que debería de existir. No podemos ver que esa línea atraviesa a todos esos datos.

Pero bueno, fue nuestra primera aproximación. Vamos a ver si otros modelos se pueden aproximar un poco más. Ahora, vamos a probar con una SVR. La SVR la importamos desde la misma librería sklearn, y le decimos la SVR ocupada para esta regresión tiene un kernel del tipo "rbf" y un gamma automático. Se entrena a través de los conjuntos de X_train y las y_train, y con eso vamos a predecir las "y" de entrenamiento y predecir las "y" de prueba. Las graficamos también.

Y aquí tenemos otro modelo de regresión hecho por la SVR. Todavía podemos observar que los datos no se aparentan al modelo adecuado, pero el error medio cuadrático, que antes era de 1.05, ahora es de 1.0061, un poco más pequeño, entonces vamos por buen camino. Vamos a probar ahora con un tipo de SVR diferente que ocupa un kernel llamado polinomio (poly).

De la misma manera SVR viene de la librería sklearn y le decimos: ahora tu kernel va a ser poli, y eso lo vamos a grabar en svr_poly, que es una variable definida por nosotros, vamos a entrenar ese

conjunto con los mismos datos...los datos de entrenamiento no han cambiado en ninguno de los tres casos y vamos a ver si lo que devuelve es un poquito más aproximado.

Lo que devuelve este caso en particular tiene un poquito más de error. Este modelo no fue tan adecuado. Quizá sea por la complejidad de los datos, quizás sea porque ese no fue el modelo adecuado, pero sigue sin satisfacer nuestra necesidad.

Vamos a ver ahora un árbol de decisión. Igual, de sklearn.tree vamos a importar ahora la regresión por árboles de decisión. De la misma manera en la variable tree definida por nosotros vamos a hacer un modelo de regresión a través de los árboles, los parámetros que ocupas son: la profundidad máxima del árbol y el número mínimo de hojas que tienen esas ramas, que el parámetro... el primero es de 12 en este caso de nuestra experimentación, el de las hojas desde 50, pero estos parámetros son un poco permisibles a la experimentación. Ustedes pueden mover a otros parámetros, otros diferentes, pero ahora sigan con esos.

Vamos a reutilizar nuestra función plot para graficar y el error fue aún menor, pero sigue sin ser exactamente lo que necesitamos. ¿Qué está pasado? Si recuerdan el conjunto de datos con el que estamos trabajando es un dato que tiene 22 tipos de variables diferentes, 22 columnas. A lo mejor las 22 no son igual de valiosas, o a lo mejor las 22 no son igual de importantes o necesarias de graficar.

En secciones anteriores hemos hablado de la importancia de seleccionar correctamente las variables que se van a ocupar. Vamos a llevar eso a la práctica en esta sección. Desde sklearn vamos a importar la selección de características y vamos a ocupar un estimador que sea una regresión lineal. A partir de ese estimador vamos a buscar cuáles son las características más importantes. Selector debe ya entender esa información por sí mismo y ahora vamos a ocupar esa selección para definir cuáles son las variables que más va a ocupar. Para eso ya no vamos a modificar directamente nuestro conjunto de entrenamiento y nuestro conjunto de prueba, sino que lo vamos a grabar en un conjunto nuevo. X_train reducida (X_train_reduced) va a grabar todas las variables del entrenamiento que son importantes de acuerdo al selector, y X_test reducida va a grabar todas las variables de entrenamiento que son importantes de acuerdo al selector del conjunto de prueba. y vamos a ver cómo queda ahora.

Ahora, ya no nos queda un conjunto de 22 columnas, sino que nos queda un conjunto de ocho columnas que son importantes. Según la paquetería de sklearn después de manipular los datos a través de un modelo de regresión lineal, las ocho variables más importantes son las que están en pantalla, Amina Flow, Ore Pulp pH, Ore Pulp Density, Flotation Column 01, Flotation Column 03, Flotation Column 04,, 06 y 05.

Bueno, ustedes las pueden ver en pantalla. Cada una de esas representan más información que el resto del conjunto de datos. Ahora si hacemos un modelo de predicción solamente con estos nuevos datos es de esperarse que sea más parecido al modelo que estamos esperando o el modelo que representa mejor nuestros datos. Vamos a ver cómo nos queda nuestra gráfica. Aún sigue sin ser completamente adecuada a nuestras necesidades pero, no se preocupen, ya experimentamos con varios modelos, siguen sin ser adecuados, pero ¿por qué estamos experimentándolos uno por uno nosotros?

Debe haber un método un poco más general que haga que esta tarea sea más sencilla, más fácil y más tranquila para ustedes. En la próxima sección vamos a hablar de lo que se conoce como método de ensamble, donde los métodos de ensamble van a ser una mezcla de múltiples métodos de

regresión para tratar de encontrar un método un poco más complejo o que agarre partes de diferentes métodos para hacer un modelo que se parezca más a la realidad de nuestros datos.

## Práctica sobre Métodos de Regresión y Ensamble

En la sección anterior vimos cómo con un problema nuevo tenemos que estar probando constantemente diferentes métodos de regresión para poder determinar cuál es el más adecuado para el problema.

Difícilmente al ver el problema nuevo vamos a poder determinar cuál es el mejor método, por consiguiente esto se vuelve un proceso algo tedioso de prueba y error que es algo que tenemos que hacer.

Una vez que ya acabamos todas nuestras posibilidades y ya no tenemos otros métodos para seguir probando y no pudo mejorar el problema, una alternativa muy eficiente es utilizar métodos de ensamble . Estos nos proporcionan una manera de utilizar modelos de forma colaborativa a medida de que van a mejorar el error en un sistema similar a un sistema de votación donde el conjunto de modelos va a tener una mayor precisión que los modelos individuales.

Entonces, para empezar a ver cómo se utilizan estos métodos de ensambles vamos a ir a nuestro cuaderno de Jupyter llamado "Métodos de regresión y ensamble".

Los métodos de ensamble de modelos utilizan múltiples algoritmos de aprendizaje para obtener un mejor rendimiento predictivo que el que se podría obtener de cualquiera los algoritmos de manera individual. Hay algunas categorías o distinciones entre los diferentes modelos de ensamble. Nosotros no vamos a entrar en el detalle pero hay algunos que trabajan de forma… de manera, como les decía, un sistema de votación o uno seguido del siguiente, es decir, el resultado del primer modelo se pasa a un siguiente modelo, a un siguiente modelo, siempre tratando de corregir el error que tuvo el modelo anterior.

Uno de los métodos de ensambles más populares es el "bosque de regresión". Si recuerdan vimos uno llamado "árbol de regresión" la intuición nos diría que el bosque de regresión es un conjunto de árboles de regresión que trabajan de manera paralela, de forma de que los resultados de estos árboles de regresión se puedan concentrar en uno solo.

Ok, la palabra "random" viene de que cada uno de estos árboles de regresión se construyen de una manera aleatoria, no a la hora de construirlo, sino a la hora de seleccionar qué variables va a utilizar ese árbol en específico. Por ellos podemos tener un conjunto de árboles diferentes que trabajan juntos.

Afortunadamente nosotros no tenemos que programar eso. Python y sklearn ya tienen todas estas librerías preparadas y las utilizamos una manera muy similar a como usamos nuestros modelos simplificados, o individuales, de forma que nosotros podemos utilizar sklearn.ensemble, que es la librería donde se encuentran todos estos métodos, y podemos utilizar, por ejemplo, el "random forest". Al random forest le podemos dar algunos parámetros, por ejemplo, cuántos árboles individuales queremos. En este caso vamos a indicarle que 100 y lo vamos guardar en una variable rf.

Luego de rf le puedo decir: entrena, o ajusta, de una manera muy similar a los modelos individuales. Una vez que lo hacemos podemos hacer las predicciones tal y como estabamos haciendo antes predicción, y podemos imprimir.

Si lo ejecutamos va a tardar un poquito más que un árbol de regresión, porque está construyendo muchos de forma paralela, pero al final el resultado se va a construir de forma similar a que si lo haríamos con un árbol de regresión.

Ahora sí, la gráfica, aunque no es perfecta, ya se ve mucho mejor que los modelos individuales. Si se fijan los puntos azules están muy pegados a la línea roja. Los naranjas todavía no logran llegar a un grado perfecto, todavía podemos mejorarlo de alguna manera, pero la diferencia con los modelos que construimos antes es muy grande. Ahora sí.

¿Qué podríamos cambiar? Bueno, hay algunos otros algoritmos de ensamblaje. Por ejemplo, está este. GradientBoostingRegressor que también construye árboles de regresión, pero en lugar de generarlos de manera paralela los construye uno seguido del otro. Se ejecuta de una manera muy similar y nos arroja a este resultado donde los puntos azules se acercan todavía más y los naranjas parece que tienen una tendencia a seguir la línea roja.

O al menos en mayor medida a nuestro algoritmo anterior. Entonces, pareciera ser que este modelo que acabamos de crear está funcionando mejor que todos los que vamos quedaban con anterioridad, pero, como les decía, todavía hay espacio de mejora.

¿Qué podríamos hacer? Bueno, una alternativa es empezar a hacer ingeniería de datos de manera que estamos creando datos artificiales a partir de los que ya tenemos. Por ejemplo, seguramente en el proceso de minado el nivel de contaminación que tenemos en nuestra muestra no se produce de manera instantánea con las medidas que tomamos. Sino que es un proceso que tomó tiempo y se puede reflejar a través de varias medidas.

Es decir, si nosotros encontramos la manera de introducir datos históricos a nuestro conjunto de datos podríamos mejorar la predicción. Este conjunto de eventos sobre el tiempo se maneja mucho sobre series de tiempo, y de hecho es un tema que vamos a ver en la siguiente semana. La idea que les quiero reflejar es: ustedes pueden construir con un problema modelos individuales de manera que vean si la predicción es lo suficientemente buena.

Si no, pueden hacer ingeniería de nuevas variables de manera que puedan tal vez deducir nuevas variables artificiales, y si no, pueden utilizar métodos de ensamble. Esto sobre todo va a ser necesario cuando la complejidad de sus problemas se empieza a escalar. De hecho, en los concursos de Machine Learning los métodos de ensambles son los que dominan las competencias.

Entonces, es algo muy importante que aprendan a dominar ya a futuro. Esto lo manejamos de una manera muy superficial para que sepan qué es lo que es y que sepan que en Python lo pueden utilizar rápidamente, pero, yo les recomendaría que primero utilicen las técnicas básicas y luego pasen a los métodos de ensamble.

# Semana 4

## Introducción

Esta semana se centrará en que aprendan de los algoritmos de agrupamiento, y qué son las series de tiempo. De la misma forma que en semanas anteriores, empezaremos por definir cada uno de los elementos relacionados, y posteriormente; utilizaremos Python para desarrollar ejercicios donde podamos aterrizar los nuevos conceptos.

Los algoritmos de agrupamiento, comúnmente conocidos por su nombre en inglés; algoritmo de "Clustering", consisten en procedimientos que buscan agrupar de forma automatizada diferentes elementos que tengan características semejantes entre ellos. Esta semana trabajaremos con el algoritmo de clustering "Kmeans", debido a su simplicidad y gran popularidad, el cual utilizaremos para agrupar a un conjunto de usuarios de tarjetas de crédito, en diferentes clusters, a partir de sus diferentes hábitos crediticios; y el manejo de su tarjeta.

Después, exploraremos el término de series de tiempo, y para ejemplificarlo; utilizaremos una serie de tiempo, tomando las acciones de una empresa de la bolsa mexicana de valores, analizaremos su historial y la información que tenga actualmente, para después poder predecir los valores que tendrá en momentos futuros. El ejercicio de esta semana está enfocado en presentarte un método de predicción basado en técnicas de Machine Learning.

## Preparando los datos para Clustering

En esta sección vamos a empezar a preparar todos los datos que vamos a ocupar para llevar a la práctica el proceso de clustering. A lo largo este video te voy recordando la importancia de por qué se debe preparar los datos correctamente y más adelante hablaremos de qué es clustering, de una estrategia de clustering que se llama k-means, pero toda esa información está en la misma libreta. En tu carpeta de cuadernos tienes una que se llama "Clustering y K-Means".

Nuestra libreta parte de una base de datos que se encuentra en el sitio Kaggle. No te preocupes, también está descargada en tu carpeta de data que está en la misma carpeta que tus libretas, pero si quieres saber un poco más de este conjunto de datos accede a la liga y ahí está la información. De todas formas aquí hay una breve descripción de todas las columnas que componen este data set. Pero bueno, vamos a empezar a reproducirlo.

Lo primero, igual que las veces anteriores, vamos a importar Pandas, vamos a importar NumPy, vamos a importar matplotlib. Pandas para manejar nuestros archivos, matplotlib para graficarlos con las mismas convenciones que lo hemos hecho en otros videos: "pd" para Pandas, "np" para NumPy y "plt" para matplotlib.

Entonces, le decimos: nuestro data frame va a ser igual a pd, de pandas punto, lee el archivo csv y aquí está la dirección del archivo. Vamos a darle ejecutar. Hasta aquí lo único que hemos hecho es cargar las tres paqueterías que vamos a ocupar. Le indicamos que vamos a graficar en esta libreta y leímos el archivo. Pero bueno, nuestro archivo tiene las siguientes columnas. Aquí hay 18 columnas diferentes que van desde la identificación de el ID que identifica cada una de las tarjetas de crédito.

Esta es una base de datos de información de tarjetas de crédito. El objetivo detrás de todo esto es ver, a partir de las 17 restantes variables que están aquí, tratar de agrupar los... a los clientes

dependiendo sus hábitos de uso de la tarjeta. Podemos leer cada uno de ellos, por ejemplo, voy a leer un par: balance, es decir la cantidad de dinero que les queda en su cuenta después de hacer compras. Número de compras, número de compras en una sola exhibición, si pagan por adelantado en algún servicio, entre otras.

Pero bueno, vamos a empezar a preparar los datos. Para poder preparar los datos necesitamos observarlos primero para saber qué es lo que vamos a preparar. Acuérdese que la instrucción df.head nos va a dar los primeros cinco elementos de este data frame. Aquí tenemos cinco elementos, está el ID y están cada una de sus diferentes columnas que lo componen. Uno puede moverse hacia un lado o hacia el otro para ver las variables.

Recuerden que si aquí le especifican en un número, por ejemplo 10, mostrará 10 elementos en lugar de 5. Ahora sí, vamos a darle "describe" para tratar de encontrar un poquito más de información.

Al darle "describe" nos dice por aquí que tenemos 8,950 elementos en la columna de balance, 8,950 en la frecuencia de balance, y si uno es observador y está recorriendo esto llegan a datos como este. Nos dice que en el límite de crédito hay 8,949 elementos. Es contrastante contra los 8,950 elementos que tienen todas las demás columnas. Quizás esa columna está incompleta, quizás falta un dato. Entonces, nuestro primer objetivo va a ser identificar esos espacios en blanco. Pero también tenemos esta, que en el pago mínimo hay sólo 8,637 registros. Bastante menos que el... que lo que nosotros queremos. Bueno, vamos a ver cómo se ven esos.

Aquí vamos a contabilizar cuáles son los elementos que tienen vacíos y vamos a ver identificar los del que más vacíos tenga, al que menor vacíos tenga. Al menos los tres primeros para ver. Aquí vemos que el que más vacíos tiene es la columna de pagos mínimos, que son 313, la columna límite de crédito que es un elemento vacío, y esta columna que tiene cero elementos.

Bueno, Python lo que lo que intentó decirnos es que en particular sólo dos elementos tiene faltantes, el tercer elemento simplemente nos mostró el último de la lista.

Si le hubiéramos pedido, por ejemplo, 5 nos hubieran mostrado pagos mínimos, límite de crédito, tenure, y nos hubiera mostrado esta columna y esta columna, las últimas dos en sucesión. Pero bueno, vamos a preguntarle ahora de qué tipo son cada uno de los valores que están aquí. Entonces, tenemos por aquí el tipo de cada una de las columnas, nuestro CUST_ID es un objeto que está asociado a la cadena de caracteres que definen la identificación de ese... de esa tarjeta.

Todas éstas son variables que son flotantes. Sino recuerden qué es una variable flotante o si no lo conocen son variables que tienen punto decimal. En particular estas dos variables son enteras, es decir, un número sin parte decimal.

De la misma forma estas son flotantes y esta última son enteras. Bueno, vamos a tratar de ahora nuestro proceso de limpieza homologar que todo nuestro data frame sea del mismo tipo. Para eso tenemos múltiples opciones. La primera y más importante: nuestro ID a la hora de estar trabajando y tratar de que la información sea lo más privado posible, vamos a tirarla de nuestra data frame de tal manera que todas las decisiones que hagamos sean si saber sobre qué clientes estamos afectando. Entonces, la instrucción "drop" lo que hace es, de nuestro data frame, remueve la columna con este encabezado. Entonces, vamos a remover la columna de customer id, de tal forma que todos nuestros datos ahora sí sean solamente datos y no sea información que puede ser relacionada a una persona, sino a sus hábitos simplemente.

Y esta instrucción de aquí nos dice: todo lo que está en tu data frame cambiarlo al tipo flotante de tal manera que todos, aunque sean enteros, va a rellenar con una parte decimal, por ejemplo, si tienes el número 5 entero lo va a cambiar a 5.0. Ustedes podrán pensar "bueno, ¿para qué me sirve el punto cero?" Simplemente va a ser para facilitar al algoritmo que trabaje siempre con el mismo tipo de valores. Todos los flotantes para hacerlo más sencillo.

Y esta última instrucción dice: rellenar los na. Hay múltiples métodos para rellenar los na: se les puede dar un valor arbitrario, se les puede dar el valor mínimo encontrado en la tabla, el valor máximo, un valor promediado de varias... esta decisión solamente queda en las manos del investigador de datos.

Nosotros para este video hemos decidido utilizar este método, ffill. ffill lo que hace es, si tienes un valor previo que sí exista, lo copia en la columna y rellena el vacío con el valor de la de encima. Simplemente para facilitar los cálculos. Entonces, el método de decisión... están invitados ustedes a probar distintos, en la documentación hay más, pero ahora vamos a trabajar con este. Le damos "ejecutar", y ahora vamos a mostrar el tipo.

El tipo de cada una de las columnas, todos son flotantes, están homologados, ahora vamos, a diferencia de como estaban antes, que eran 1's enteros, 1's flotantes, ahora todos son flotantes... vamos a contabilizar los que sean vacíos.

Gracias a la instrucción "fillna" ya no existe ningún vacío, ninguna columna tiene más de 0 vacíos, y eso se debe a que... acuérdense que lo rellenamos con el dato que estaba en el registro del campo anterior y ahora vamos a graficarlo en una gráfica de caja, ¿para qué? Para poder darnos una idea de cómo están nuestros datos.

Si nosotros graficamos nos queda esta imagen, y esta imagen se ve poco representativa, ¿Por qué? Fíjense, hasta acá arriba tenemos el número 80,000 y abajo 0, y esta caja está muy pegada a 0, esta caja también, está también, excepto esta, esta caja está muy despegada. Cada una de estas variables tiene una escala diferente. Si todas las graficamos dentro de la misma escala pues se verían poco representativo o se verían mal como en este caso.

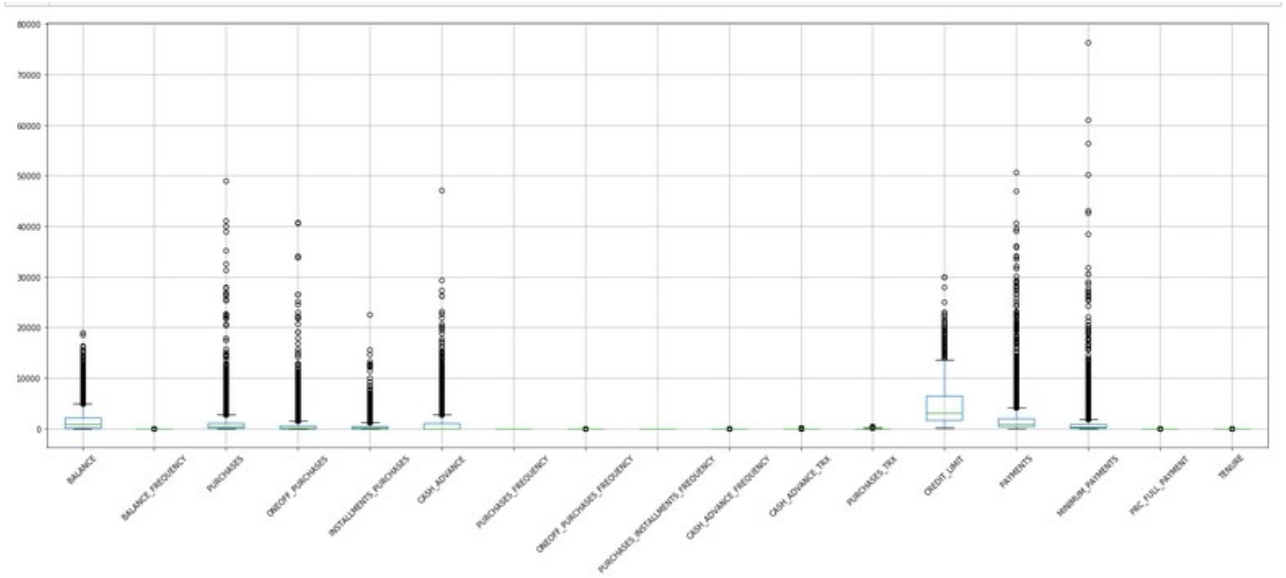


*Figura 31: Boxplot 1*

Tenemos alternativas, la primera alternativa es ocupar una escala logarítmica. Aquí la diferencia está en que mandamos a imprimir directamente el box plot del data frame y acá, ocupando NumPy, pudimos ocupar la opción "logaritmo". Para los que saben de matemáticas, el logaritmo de cero no está definido. Entonces, si alguno de nuestros datos de nuestro data frame es cero, a la hora de graficar nos va a marcar ahí una función que, con error, nos va a decir: hay un error matemático que impide que yo grafique eso. Por eso ven esa consideración. No estamos sacando directamente el logaritmo del data frame, sino que estamos haciendo uno más el valor del data frame. Vamos a ver cómo se vería graficado esto.

```
# Graficar con escala logaritmica para mejor visualizacion
np.log(1+df).boxplot(figsize=(28,10))
plt.xticks(rotation=45);
```

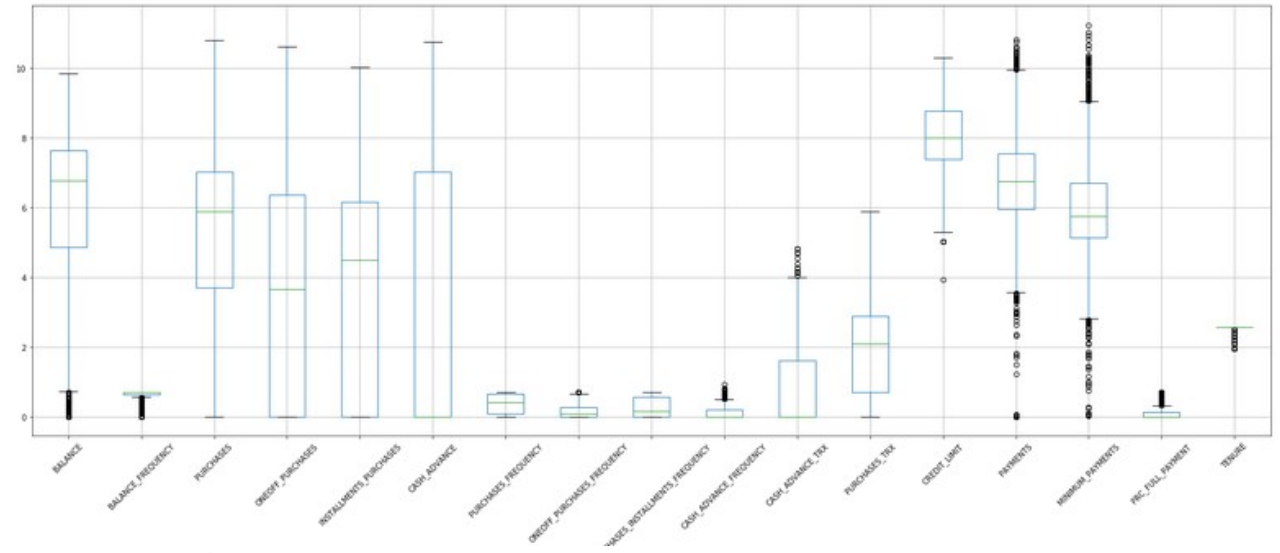


*Figura 32: Boxplot 2*

Ya nos da cajas adecuadas a un valor de 0 a 10 donde ya puedes más o menos identificar, ahora sí, en la misma escala todas las variables. Pero bueno, vamos a seguir trabajando con estos datos, y para poder trabajar los en el algoritmo de clustering que vamos a usar más adelante, vamos a respaldar nuestra data en una nueva variable, pero esta variable va a tener la característica que, en lugar de ser datos representados en términos de la variable que están viviendo, van a ser datos normalizados, y todos esos datos van a ser pasados a una escala de -1 a 1.

En secciones anteriores ocupamos otro tipo de normalización que era las MinMax que normalizaba de 0 a 1. Ahora, en particular por cómo funciona el algoritmo de clustering que vamos a estar trabajando, vamos a ocupar una normalización de -1 a 1. Esa normalización la vamos a encontrar en esta sección. Vamos a decirle: de... de la paquetería a sklearn.preprocessing vamos a importar el estándar scaler y el normalize.

Con el estándar scaler vamos a estandarizar la data, vamos a generar una variable que se llama scaler, y vamos a ahí... asignarle el estándar scaler. Y a scaler vamos a hacer una transformación de nuestros datos del data frame y después vamos a normalizar esos datos.

Vamos a reproducirlo...y ahora vamos a ver cómo se ve ese nuevo data frame. Entonces, nuestro data frame, que ahora tiene valores entre 0... entre -1 y 1, representan los mismos datos que teníamos acá arriba... estos....estos de aquí... pero ahora de una forma normalizada.

En nuestra siguiente sección explicaremos qué es el clustering y que ya va a ser posible hacerlo gracias a que tenemos nuestros datos limpios de vacíos, estandarizados en el tipo de datos que son, hemos y estandarizados los datos y los hemos normalizado utilizando estas instrucciones que tenemos aquí. Bueno, en la siguiente sección continuaremos utilizando la misma libreta y veremos lo que es clustering.

## ¿Qué es Clustering?

Continuaremos utilizando la libreta en el momento en el que la dejamos de la sección anterior. El objetivo de este video es explicar qué es clustering. Tenemos una parte de teoría, igual que en las secciones pasadas, en la que vamos a explicar brevemente qué es clustering, y luego vamos a explicarles un método en particular de clustering, pero eso será cosa de la siguiente sección.

Antes que nada, ¿qué es clustering? Clustering es una técnica que busca identificar de manera automática, sin que nosotros le estemos retroalimentando si está bien o está mal, diferentes agrupaciones. La palabra cluster es una palabra en inglés, la tenemos en el español y podría ser igual cluster, pero acentuada, que se define como agrupaciones, clusteres es el plural en español, pero van a oír que a lo largo de esta sección y posteriores vamos a usar el término clusters, o clustering, que son términos del inglés pero son con los que se conocen comúnmente más en el área, porque métodos de agrupación no es tan común que lo escuchen como científico de datos.

Pero bueno, dejando de lado esa nota vamos a continuar. Entonces, el clustering es una técnica que busca identificar de manera automática agrupaciones de elementos dependiendo de la similitud que existan entre ellos. Todos los elementos que se parezcan los van a poner en un grupo, los otros elementos que se parezcan en otro grupo.

La meta es que el algoritmo encuentre grupos donde la similitud entre sus miembros sea lo mejor posible, o lo mayor posible, y la similitud con miembros de otro grupo sea lo mínimo posible. En nuestra pantalla tenemos una especie de esquemita. En este esquemita, los datos que están ahí regados por toda esa gráfica, han sido separados en cuatro clusters diferentes: un cluster amarillo, un cluster morado, un cluster rojo y un cluster azul. Cada color representa conjuntos de datos en los que datos de similares características se han agrupado, y hay un punto verde marcado en cada uno de esos grupos.

Este algoritmo del cluster que ocupamos para seleccionar estos clusters parten de algo que se llama, centroides, pero eso lo vamos a ver cuando ya entremos más a detalle. Existen múltiples algoritmos de clustering, pero a continuación se presenta un pequeño esquema donde se observan cuatro clasificaciones diferentes de algoritmos de clustering. Tenemos algoritmos de clustering basados en centroides, basados en densidad, basados en distribución, basados en jerarquía.

Nuestro curso escapa de explicarles cada uno de estos métodos. Vamos a agarrar uno muy popular que se llama k-means. K-means corresponde al tipo basado en centroides. En nuestra libreta ustedes podrán observar una liga que está aquí abajo, que developers.googlemachinelearningclusteringclusteringalgorithms que se les invita a ustedes

como actividad a que la lean. En esa liga encontrarán diferentes tipos de algoritmos de clustering para su información. En nuestra siguiente sección continuaremos con esta información pero tomando en cuenta el algoritmo de k-means, desarrollando un ejemplo y después, en una sección posterior, ya lo llevaremos al código.

## K-means

Continuaremos usando nuestra misma libreta y más abajo va a salir una información de un algoritmo de clustering basado en centroides muy popular que se llama k-means. Pero bueno, vamos a ir a esta información y vamos a explicarla juntos.

Nuestro algoritmo de k-means, como ya les comenté, es un algoritmo de clustering. Es un algoritmo que agrupa diversos elementos en "k" grupos. Nuestro primer parámetro de nuestro algoritmo es el número de grupos en el que yo deseo agrupar mis datos. Si yo quiero... si yo defino que "k" es 3 voy a tener 3 conjuntos de datos, si defino que van a ser 5, 5 conjuntos de datos y así sucesivamente.

¿Cómo los va a agrupar? Bueno, basándose en la similitud de características que existen en estos elementos. El agrupamiento se realiza a través de un proceso iterativo, los procesos iterativos son un proceso en el que se desarrolla todos los pasos una vez, después se repiten, después se repiten y se repiten hasta que el investigador o el algoritmo decidan detenerlo.

Hay diferentes formas de detener este tipo de algoritmos, uno puede definir desde el principio, bueno, quiero que mi máximo de repeticiones sean 15, entonces, al llegar a la 15 integración el algoritmo se va a detener en donde se haya quedado o le puedes decir, bueno, quiero que si durante cinco iteraciones no has modificado mucho tu comportamiento o te has estancado, ahí te detengas dándole un poquito más de flexibilidad para que el algoritmo determine su final por medio de su propia ejecución.

Pero bueno, ¿cuáles son los pasos de ese algoritmo? El primer paso: se van a asignar centroides o puntos que son los que estaban marcados en nuestra imagen anterior de clustering como puntos verdes, ¿cuántos centroides se van a asignar? Tantos como grupos desee tener. Si yo quiero tener cinco grupos voy a designar cinco centroides, y ¿cómo se van a asignar? Bueno, se van a asignar al principio de una manera completamente arbitraria, es decir, los vamos a poner simplemente en donde nosotros decidamos.

Después, el paso dos: vamos a crear conjuntos tomando en cuenta cuáles son los elementos que están más cerca a cada uno de sus centrales y después vamos a actualizar los centroides en caso de que sea necesario.

Bueno, el paso dos y el paso tres son los pasos que se van a repetir de forma iterativa: vamos a crear conjuntos, vamos a actualizar el lugar de los clusters, creamos los nuevos conjuntos, actualizamos, creamos los nuevos, actualizamos y así sucesivamente.

Nuestro siguiente procedimiento, ya que definimos los grupos, es intentar mover los centroides. Estos centroides se intentan mover a partir de minimizar la distancia al cuadrado que separa la suma de los elementos del centroide. Ahora simplemente quédense con la idea conceptual de que el centroide se va a mover para intentar abarcar la mayor cantidad de puntos con el mínimo de distancia que los separen entre ellos. Entonces, este centroide se va a mover

¿Qué pasaría una vez que ya están centrados? Los puntos con los que están más cercanos van a ser diferentes.

Básicamente esa es la idea que existe detrás de k-means, y en nuestra siguiente sección lo llevaremos a la práctica en nuestra libreta de Python.

## Usando K-means

Bueno, ya hemos hablado de la importancia de la preparación de los datos, ya hablamos de qué es un algoritmo de clustering en grandes términos incluso ya ejemplificamos visualmente qué es un algoritmo de k-means perteneciente a la familia de algoritmos de clusterings basados en centroides.

Bueno, vamos a utilizar los datos que preparamos previamente acerca de los usuarios de tarjetas de crédito y de su uso para esta práctica. En nuestra misma libreta, más arriba, acuérdense que están esos datos y esos datos los preparamos quitándoles los enteros, quitándole los identificadores, rellenando los blancos, bueno, esos datos.

Entonces, lo primero que vamos a hacer es, de la paquetería sklearn.cluster vamos a importar KMeans. Entonces, lo ejecutamos, simplemente en este entorno ya podemos usar k-means, y vamos a asignar un número de dos clusters, vamos a empezar pequeño. ¿Por qué dos clusters? Porque al agrupar en uno solo pues sería todo el conjunto de datos. El mínimo real, aunque se pueda uno, pues son dos para que haya dos elementos diferentes.

Y, ¿qué va a hacer? Bueno, vamos a ejecutarlo. Aquí ya le dijimos al algoritmo de k-means el parámetro más importante que va a tener, el número de clusters, y con qué datos lo va a ser, con nuestros datos normalizados.

Bueno, si yo gráfico esto con qué tengo por aquí. Bueno, antes de graficar necesito, de la paquetería sklearn.decomposision, importar a PCA. PCA va a ser un método que vamos a ocupar, recuerden que el conjunto de datos tenía 17 variables, las 17 variables si las quisiéramos graficar sería un espacio de 17 dimensiones, y realmente no creo que hay una forma de entender sencillamente 17 dimensiones. Para eso vamos a ocupar un método de decomposición en el que las 17 variables diferentes las va a intentar emular, o intentar resumir en dos variables.

Esas dos variables generadas van a ser como que un reflejo de la importancia de las 17 variables como tal y son las que vamos a poder graficar. Nuestras dos variables van a surgir de hacer la transformación de los conjuntos normalizados y eso lo que vamos a graficar.

Entonces, tenemos por aquí nuestras dos variables, nuestras dos variables hemos mostrado las cinco primeros datos. Para el conjunto 1 son todos estos datos para el conjunto 2 son todos estos. Este es un... recuerden, como un reflejo de las 17 variables y ¿qué vamos a ver? Bueno, si yo grafico por aquí tenemos este scatter donde todo nuestro conjunto de puntos en este plano de las 17 variables comprimidas en dos han sido separados en el conjunto morado del lado izquierdo y en el conjunto amarillo del lado derecho.

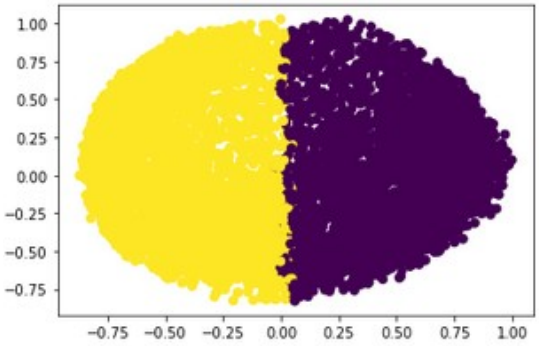


*Figura 33: PCA scatter plot*

¿Cómo sé cuál es el número de clusters que debo usar? El número clusters es nuestro parámetro más importante. La primera forma que les vamos a presentar para decidir cuántos clusters vamos a usar es un método que se llama el "método del codo", más conocido por su nombre en inglés "elbow method".

Vamos a hacer referencia a ese nombre en inglés porque es mucho más fácil encontrar información del nombre en inglés que de "el método del codo". Entonces, elbow method, ¿cómo lo hace? Bueno, este método parte de la suma de las distancias al cuadrado, y para cada "k" en el rango del 1 al 15 va a intentar probar cuál sería la suma de distancias para un conjunto que tenga un solo cluster: k=1, dos clusters k=2, tres clusters k=3 y así sucesivamente hasta llegar a 15 clusters.

Recuerden que la meta es tratar de minimizar la suma de distancias acumuladas por cada uno de los clusters. Entonces esta funcioncita con el ciclo de repetición "for" va a ir calculando esas distancias, va a calcular la suma de distancias con un cluster, con dos clusters, tres clusters y así sucesivamente. Vamos a dejar que se reproduzca...y después vamos a graficarla para visualizarla.

```
Sum_of_squared_distances = []
for k in range(1,15):
    km = KMeans(n_clusters=k)
    km = km.fit(normalized_df)
    Sum_of_squared_distances.append(km.inertia_)
```

```
plt.plot(Sum_of_squared_distances);
```

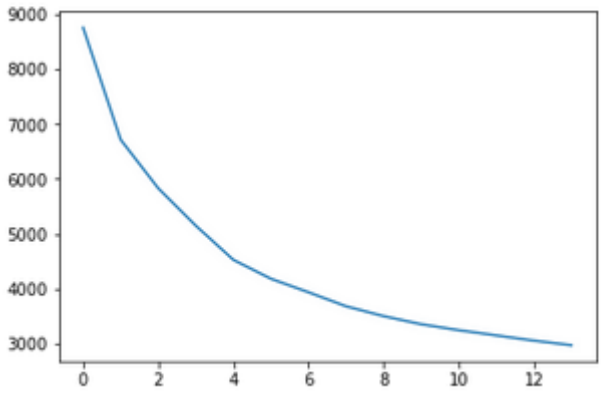


*Figura 34: Método del codo*

Entonces, con un solo cluster la distancia fue muy alta, está aquí cerca de este 9000, con dos estamos cerca de estos 5,600, 5,700, con cuatro clusters estamos cerca de 4,000 y así sucesivamente.

Nuestro método se llama método del codo porque si uno pone esta gráfica en donde se hace una especie de codo o una especie de de división en que los datos iban bajando con una velocidad y de la nada después de ese punto empiezan a bajar mucho más rápido, supuestamente los clusters que representen este punto son el número de clusters que vamos a ocupar, pero este es un método que parece ser un poco arbitrario.

Muchos investigadores critican mucho que este método es un tanto arbitrario y está sujeto a la preferencia o delimitación del investigador. Yo podría decir: ok, yo veo que esto cambia a partir del 4, pero ustedes a lo mejor pueden ver que la diferencia ocurre desde el 2 o desde el 5. Entonces, es ambiguo.

Pero bueno, podríamos determinar que el óptimo en el momento en el que empieza a girar esta curva está entre el 4 y el 6. ¿Qué alternativas tenemos? Hay otro método más aceptado o menos ambiguo que se llama el método de la silueta. Explicar el método de la silueta como tal se sale de los alcances de este curso, pero eso no significa que no podamos implementarlo. Simplemente significa que ustedes, como investigadores, quieren ahondar más en el tema les recomendamos que lean un poco más la definición formal de qué es el método de la silueta.

Bueno, el método la silueta lo tenemos aquí definido en esta sección y nos dice: importando la métrica (porque esto ya es importante, ya es una métrica) de la paquetería sklearn.metrics, vamos a importar el score silueta (silhouette_score). El cálculo interno que se realiza con la silueta es el que les comento, se escapa del alcance de este curso y simplemente hay que verificarlo, pero ¿qué es lo importante que ustedes deben de saber? Lo importante es que es un score que se intenta minimizar. A medida que sea más chico es más conveniente para nosotros que un score alto.

Si yo reproduzco esta parte de aquí estoy calculando para diferentes clusters cada uno ese score silueta y después voy a graficar todas esas siluetas, o más bien todos esos score silueta. ¿Cómo los voy a graficar? pues con una gráfica de barras para ver cuál es más grande que otro. Le doy reproducir y nuestra gráfica debe aparecer.

```
from sklearn.metrics import silhouette_score
silhouette_scores = []

for n_cluster in range(2, 8):
    silhouette_scores.append(
        silhouette_score(normalized_df, KMeans(n_clusters = n_cluster).fit_predict(normalized_df)))

# Plotting a bar graph to compare the results
k = range(2,8)
plt.bar(k, silhouette_scores)
plt.xlabel('Numbero de clusters', fontsize = 10)
plt.ylabel('Silhouette Score', fontsize = 10)
plt.show()
```

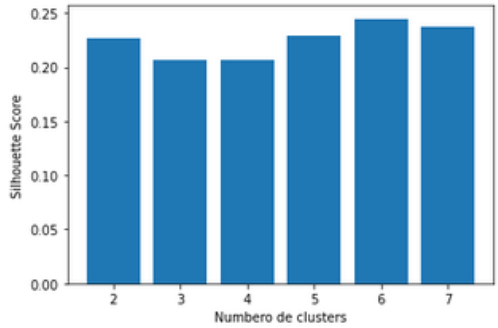


*Figura 35: Grafico de silueta*

En nuestra gráfica tenemos diferentes barras. Tenemos en particular desde el dos clusters hasta siete clusters. Nuestro valor mínimo que tenemos visualmente es el de tres y el de cuatro clusters. Según el método de la silueta usar 3 o 4 clusters separaría bien nuestro conjunto. Nosotros ahora, para respetar ambos métodos, vamos a tomar el 4, porque el 4 coincidía hacer con lo que nosotros observamos en el método del codo o del elbow, y el 4 es el mínimo del método de la silueta. Entonces, para nuestro ejercicio de k-means que va a venir a continuación vamos a ocupar cuatro diferentes centroides.

El resto del ejercicio es lo siguiente: para nuestros cuatro centroides, que definimos previamente, vamos a ocupar k-means con ese 4, vamos a ocupar nuestros datos normalizados y vamos a hacer el scatter que nos muestre esos datos. Aquí ya tenemos cuatro conjuntos en el que fueron separadas e

identificadas características de nuestro conjunto de datos. Recuerden que lo que estamos observando es una adaptación de las 17 variables diferentes adaptadas a un espacio de dos dimensiones. Pero bueno, vamos a agregar la etiqueta de cada cluster para ver nuestros datos.

Aquí ya tenemos de nuevo nuestro conjunto de datos original, ya no el normalizado. A cada uno de esos elementos le agregamos una columna "c" que depende de qué cluster corresponde. Aquí tenemos la columna de balance, de frecuencia, de compras que son las originales de nuestro problema, pero hasta el final ya tenemos esta columna. La columna que nos indica a qué cluster pertenece. En este ejemplo el elemento 0 pertenece al cluster 2, el elemento 1 pertenece al cluster 1, el elemento 2 al cluster 3, y así sucesivamente.

Nuestros datos normalizados nos sirvieron para calcular en qué cluster iban a pertenecer pero ya no nos van a servir de aquí en delante. Simplemente nos va a servir el resultado de haberlos normalizado que se lo vamos a agregar a nuestro data frame original. Por eso ven que ya estamos de vuelta trabajando con el data frame original, ya no con el normalizado. Ok, ahora, ¿de qué nos sirvió todo esto de los k-means, de los clusters y todo? Hasta ahora simplemente el algoritmo le designó un número a cada uno de nuestros elementos, pero eso a nosotros como investigadores no nos ha dado ningún dato.

Vamos a regresarnos a graficar todas las columnas de nuestro data frame, pero separando los datos del cluster 1, del cluster 2, del cluster 3 y del cuarto cluster. Importamos seaborn, como en otras ocasiones, para poder ocupar una paquetería un poco más especializada para hacer gráficas con temas y un poco más estéticas, y le vamos a decir: para cada columna en nuestro data frame, cada una de las 17 columnas, vamos a graficar el valor del data frame pero con respecto a la columna "c2. Va a haber un histograma para los elementos del cluster que estén identificados con un 0, para los elementos del cluster que estén identificados con 1, con 2 y con 3.

Vamos a ver cómo se ve. Tenemos un buen conjunto de gráficas. Ahorita las vamos a explicar de manera general. ¿Cuántas gráficas tenemos? Hacia abajo tenemos 17, hacia los lados tenemos 4, 17 por la cantidad de columnas, 4 por la cantidad de clusters, 17 por 4, 68, entonces, por aquí tenemos una gran cantidad de información con la que tenemos que empezar a trabajar para tomar algunas decisiones.

```
for col in df:
    grid= sns.FacetGrid(df, col='c')
    grid.map(plt.hist, col)
```

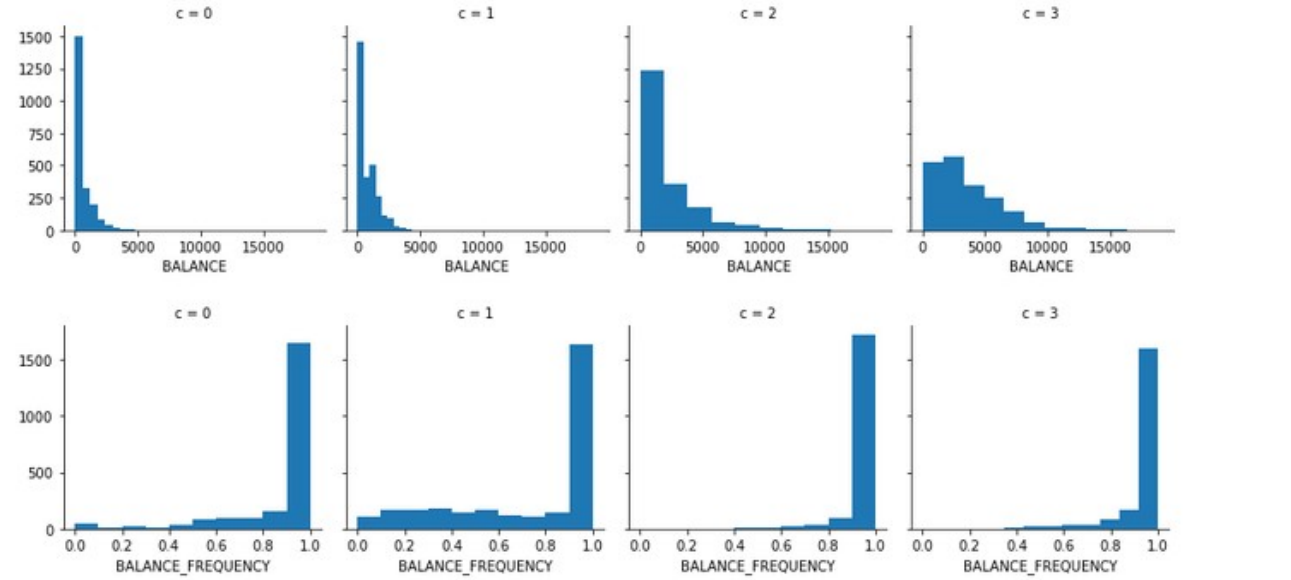


*Figura 36: Clusters para la columna Balance*

En nuestro primer reglón de gráficas estamos graficando la variable del balance y aquí podemos observar que, a medida que esto aumente, significa que más usuarios de este cluster tienen este valor de balance. Entonces, si se dan cuenta muchos usuarios de nuestro cluster 0 tienen un balance cercano a 0 dólares. Entonces, en ese cluster hay muy poquitas personas que tengan un balance alto, de hecho, las personas que tienen un balance cercano a 5,000 son muy poquitas, a diferencia de otros que, por ejemplo, aquí hay muy pocas personas que tienen un balance cercano a 10,000 pero ya está un poco más distribuida la cantidad de balance.

Entonces, en este grupo tenemos muchas personas que andan en un balance bajo, pero hay una distribución muy extensa de balances. Hay incluso personas que tienen 15,000. En esta de aquí tenemos personas que modifican su balance muy frecuentemente, en esas también, pero en este aquí no hay personas que lo modifiquen pocas veces.

Entonces, cosas de ese estilo ya empiezan a surgir, que lo que es nuestra tarea principal como científico de datos. Si a nosotros nos piden que hagamos un análisis de esto y nada más mostramos estas tablas o estas gráficas, a lo mejor nuestro cliente o socio, o con quien estamos haciendo este estudio nos va a decir: oye, y a mí para qué me sirven todos estos datos, yo solamente veo muchas gráficas. Entonces, una parte fundamental una vez que ya sabes cómo hacer las gráficas es aprender a analizarlas.

Los datos cuentan una historia y a medida que tu experiencia como científico de datos vaya incrementándose va a ser más fácil para ti ver esa historia.

Las conclusiones observadas que nosotros identificamos, que ojalá sean iguales a las tuyas, son: en el conjunto 0, en el cluster 0, son las personas que hacen gran uso de su crédito, por ello su límite de crédito es alto, no suelen comprar con frecuencia y procuran no obtener financiamiento para obtener bienes. Todo lo tratan de comprar inmediato.

Nuestro segundo cluster son las personas que no suelen utilizar con regularidad su tarjeta de crédito. Al menos no la tarjeta del crédito del banco que estamos analizando. Desconocemos si sea porque

no tienen tarjeta de crédito o por si tienen tarjetas con otros bancos, pero al menos en nuestro banco no suelen utilizar comúnmente nuestra tarjeta. El cluster 2 son las personas que utilizan frecuentemente su tarjeta de crédito, tienen muy buen balance, hacen compras frecuentes y en gran volumen, por lo tanto, también cuentan con un límite de crédito alto. Incluso pueden comprar cosas por adelantado. Si nosotros pudiéramos suponer cuáles clientes tienen más solvencia económica, hasta ahora los datos nos han revelado que son la gente que está asociada a este grupo. Pero bueno, vamos a ver cuál es el grupo 3. Son las personas que tienen malos hábitos financieros, al menos con el uso de su tarjeta de crédito del banco que estamos analizando en este momento, porque suelen usar mucho su tarjeta aún sin tener el balance o la libertad económica que lo respalde realizan muchas compras financiadas. Entonces, eso significa que muchos del dinero que van a ganar en el mes que sigue ya lo están debiendo.

Bueno, esas son conclusiones un poco rápidas, pero para qué podríamos usar esto. Ahora, imagínense que a nosotros nos invitaron a analizar estos datos un banco, y nos dice: ya identificarse los cuatro tipos de clientes que tengo. ¿Qué me puedes proponer? Ok, a lo mejor yo estoy viendo que en el grupo del cluster 1 las personas no suelen usar nuestra tarjeta de crédito. ¿Por qué? qué no les está gustando a esas personas. A lo mejor ya identificamos a esas personas, les podemos hacer una encuesta acerca de por qué no están usando nuestra tarjeta, podemos incluir cosas como: la tasa que estás ocupando de interés no te satisface, los meses que estamos presentándote de financiamiento no se acomodan a ti, necesitas un límite de crédito, o a lo mejor hay otras preguntas que nuestro equipo de captación del banco, la gente de mercadotecnia, pueden explotar para diseñar un paquete para atraer a esos clientes de nuevo a nuestro banco de tal manera que podamos ser una opción muchísimo más llamativa para ellos.

Si también identifique que en el grupo 2 son personas que usan frecuentemente su tarjeta de crédito pero que aparte amigo que tienen buen balance y aparte que hacen compras con frecuencia y todo eso, puedo decir: sabes que, esta persona tiene mucho dinero o es probable que tenga mucha holgura económica. ¿Qué tal si estas personas les ofrezco un seguro de vida?

Bueno, básicamente este es el por qué compañías que aparentan estar muy lejanas a la computación están ocupando estas técnicas de agrupamiento, o de clustering, que aparentemente requieren mucho conocimiento técnico en el área de computación y los han ido incorporando a sus procesos, pero las posibilidades no se quedan en lo poco que les hablé sino que son cientos de miles.

## ¿Qué es una serie de tiempo?

En esta sección empezaremos a trabajar ahora con series de tiempo. De igual manera que en secciones anteriores empezaremos definiendo qué es una serie de tiempo, empezaremos presentándola dentro del entorno de Python, y después haremos más ejercicios con ella.

Para estas secciones y las siguientes, preparamos una libreta que se llama "Series de Tiempo". La libreta la encontrarán en el mismo paquete de libretas, y les invito a que la abran.

Nuestra libreta empieza de la siguiente manera, empieza definiendo qué es una serie de tiempo. Vamos a leer en conjunto qué es una serie de tiempo.

Dice: usaremos las series de tiempo para predecir el cierre de una acción en la bolsa de valores mexicana. Podemos pensar en las series de tiempo como secuencias de datos que miden la misma cosa durante un período de tiempo. Podemos imaginarlo como un conjunto de datos que tiene una

fecha o una hora en la cual queda registrado qué ocurrió. Este tipo de conjunto de datos crece constantemente. Tenemos bolsas valores que están en constante crecimiento registrando los costos, existen casas inteligentes donde se crean series de tiempo a partir de la temperatura del hogar, de la energía medida, entre otras diferentes cosas.

Por lo general, en las series de tiempo, cada que un dato nuevo se genera se anexa a la serie existente. Recuerden, estos datos que están aquí son un historial, son datos que ya no pueden ser modificados pues son eventos que ocurrieron en el pasado. Además, los datos llegan ordenados por el tiempo asociado a cada registro y podrás imaginar que el tiempo es como que el eje principal en el cual estos datos se modifican.

Poder medir los sistemas presentes y guardar esta información en series de tiempo nos permite analizar el pasado, monitorear el presente e incluso intentar predecir el futuro. Cualquier serie de tiempo se puede describir en cuatro componentes básicos:

1. **el nivel**, es decir, el valor de referencia para la serie si ésta fuera una línea recta
2. **la tendencia**, el aumento o disminución de la serie a lo largo del tiempo
3. **la estacionalidad**, los patrones o ciclos a lo largo del tiempo
4. **el ruido** que son toda la variabilidad muy notoria en las observaciones.

Entonces, una serie de tiempo es una composición de estos cuatro diferentes elementos. ¿Cómo se suelen realizar las predicciones a partir de una serie de tiempo? Existen métodos clásicos basados en estadística entre los cuales resalta el método ARIMA como uno de los métodos que mejor desempeño ha tenido, y existen los métodos de Machine Learning.

En nuestro curso centraremos esta libreta y los videos asociados con los métodos de Machine Learning pero te incluiremos como material adicional para tu lectura e información algo relacionado con los métodos clásicos. Si alguno de ustedes ya tiene algún conocimiento de estadística y métodos como ARIMA, y no son desconocidos para ustedes, pues ya tendrán una idea de qué se trata esto, pero ahora quedará a ver cómo Python lo maneja para que vean cuán fácil vuelve su trabajo.

## Series de tiempo en la bolsa de valores

Bueno, vamos a ocupar una serie de tiempos tomadas la bolsa de valores de la compañía CEMEXCPO. Entonces, Yahoo Finance pone esta información a nuestra mano. Aquí está la liga para que ustedes la descarguen, pero de todas maneras nosotros ya se los incluimos en la carpeta de Data que debe haber sido bajada a la par que las libretas. De igual forma ustedes pueden accesar a la liga para que la información esté actualizada al momento en el que ustedes están haciendo la práctica o pueden seguir con el archivo que quizás tengan unos cuantos días de retraso.

Vamos a empezar a trabajar la serie de tiempo en la bolsa de valores. Para esto vamos a ocupar Pandas así como lo hemos hecho con anterioridad para poder leer el archivo separado por comas. El archivo que vamos a bajar va a ser una extensión .csv (comma-separated values) que suele poderse abrir con Excel, pero lo vamos a abrir a través de Python. Vamos a decirle que existe pyplot as plt porque vamos a estar graficando así como lo hemos hecho en ocasiones anteriores y vamos a tener la opción de matplot inline.

Entonces, vamos a reproducir esto, importamos y ahí ya debe estar en memoria nuestro archivo, vamos a indicar que vamos a ocupar matplot inline y vamos a ver los primeros datos. Vamos a centrar el análisis de nuestro data frame en los cinco primeros registros: la fecha, el valor con el que abrió en el día, el máximo valor alcanzado por la acción, el mínimo valor alcanzado y el valor de salida, hay valores adicionales pero no los tomaremos en cuenta para el análisis de este momento.

Entonces, vamos a ver qué tipos de datos estamos trabajando. Para esto tenemos la opción .dtypes. Entonces, por aquí nos sale que tenemos una fecha, que es un objeto porque es un texto numérico, tenemos valores flotantes con punto decimal para todos los demás registros, y para poderlo trabajar vamos a tomar de Pandas la función to_datetime para cambiar el data en una representación que la computadora entienda como una fecha. Vamos a ver qué pasa con esa instrucción. Fíjense como ahora el date ya lo tiene marcado como la fecha. Entonces, internamente ya lo va a poder manejar como una serie de tiempo, y le decimos: el index va a ser con una frecuencia diaria, la función asfreq es donde nosotros definimos la frecuencia con la que van a pasar los eventos, por ahí la documentación de esta función nos dice que cuando ocupamos la letra "d" va a ser una frecuencia diaria. Vamos a imprimirlo y vean la principal diferencia.

Aquí en nuestro archivo original teníamos el día 3, el día 4, el 5, el 6, el 7 y el 10. ¿Por qué se dio ese brinco entre 7 y 10? Un fin de semana se atravesó de tal forma que el día 8 y 9 se descansó. Pero bueno, en nuestro, caso como es una frecuencia diaria, al ocupar la función asfreq, nos agregó el día 8 y el día 9, pero como es de esperarse, como es un día que no se registró, todos los campos van a estar en blanco van a estar vacíos.

Ahora, nosotros no podemos trabajar con esos vacíos. En anteriores videos hablabamos de la necesidad de preparar los datos limpiando tanto datos que sean atípicos, tanto datos como que sean de tipos diferentes o datos vacíos. Nosotros en este video vamos a ocupar el método fill vacíos (fillna) y le vamos a decir: vas a ocupar este método ffill. ffill lo que hace es: el valor vacío, por ejemplo este que estoy señalando en la pantalla, lo va a rellenar con el valor anterior de tal forma que este vacío lo va a sustituir con este 8.11357, este vacío con 8.16428 y así sucesivamente.

Entonces, vamos a mostrarlo, y los vacíos que teníamos los rellenamos con los valores del día anterior. Podrán decir: a lo mejor estás copiando un número muy bueno o un número muy malo y eso nos va a dar una impresión equivocada de la información de la bolsa de valores. Algorítmicamente hace muchísimo más ruido los vacíos que estas repeticiones de valores. Entonces, simplemente es una manera de que la computadora pueda procesar estos datos.

Cuando les hable de este archivo les comenté que ustedes podrían usar el que nosotros bajamos y les adjuntamos en la carpeta de Data, o que ustedes podían bajar el archivo con las fechas más actualizadas. Para evitar discrepancias entre lo que ustedes pueden ver en la computadora y nosotros estamos haciendo ahora en nuestra libreta nosotros vamos a definir nuestro data frame como los datos que están entre el mes de julio del 2016 hasta el mes de julio del 2018. ¿Cómo? Bueno, pues cortando el data frame solamente para los registros que tengan en esta parte.

Si nosotros imprimimos la cabeza de esta sección van a ver que las fechas cambió de ser el año 2000 a ser el año 2016. Ustedes podrían jugar con estas fechas si sus intereses son otros meses diferentes u otros años diferentes pero bueno al menos ahora para que podamos homologar lo que ustedes están viendo en pantalla con lo que nosotros estamos haciendo definimos esas fechas. Vamos a graficarlo y aquí tenemos nuestros datos.

Estos son la forma en la que ha ido modificándose el cierre, porque estamos graficando el close, de los valores de la serie de tiempos de la compañía CEMEXCPO a través de todos estos meses. Podemos observar que hay meses que cerró muy bien cerca del 2017, hay meses que cerró relativamente bien, pero hubo meses, como en este del 2016 o este del 2018, en el que el precio de cierre de la bolsa fue muy bajo.

Más adelante vamos a ocupar en la siguiente sección esta información para hacer predicciones a futuro a fechas que aún no tenemos registros.

## Haciendo predicciones

Vamos a continuar con lo que hicimos en secciones anteriores. Ya preparamos la información de nuestra serie de tiempo, y ya la dejamos lista para poder hacer predicciones pero ahora falta manipulara un poco más, pero para ello vamos a contextualizar un poco.

Supongamos que queremos hacer una predicción y nos dicen ¿cuál va a ser el valor de cierre para el día viernes? Bueno, si nosotros nada más decimos "bueno, el viernes…" pues no tenemos algún antecedente para poder decir "pues va a ser 10 o va a ser 8, verdad.

Entonces, generalmente necesitamos algún tipo de preparación o contextualización para poder decir "va a ser cierto valor". Comúnmente sería los valores de los días anteriores: un día antes, dos días antes, tres días antes, y con eso ya nosotros podemos suponer cuál va a ser el valor de cierre.

Ok, lo mismo pasa para cuando hablamos de Machine Learning. Lo que queremos hacer es, si vemos nuestra grafica en el cuaderno, tenemos un valor actual. El valor actual sería el valor siguiente que queremos predecir, digamos que sea el día de mañana. Lo que vamos a querer preparar es tomar el día anterior, dos días anteriores, tres días anteriores y transformar esto de alguna forma para que nos dé valores que nos puedan servir como variables independientes. ¿Para qué? Para poder predecir nuestra variable dependiente que sería la siguiente predicción.

Este fenómeno de estar tomando valores anteriores y manipularlos de alguna forma se conoce como ventanas de tiempo. También se conocen como ventanas de tiempo deslizantes, pero es más simple llamarlo ventanas de tiempo. Esto, si lo reflejamos a lo que conocíamos antes como regresión, podemos ver que ya toma cierta semejanza. Tenemos algunas variables que podemos tomar como "x" que son las medidas anteriores, y una variable "y" que sería la que queremos predecir. Entonces, efectivamente estamos transformando una serie de tiempo a un problema de aprendizaje supervisado. Es, más en específico, un problema de regresión.

Ok. ¿Para esto cómo le vamos a hacer? Bueno, vamos a regresar a nuestro cuaderno de Jupter y vamos a crear dos variables nuevas. Una, vamos a decir cuántos datos a futuro queremos reservar para hacer pruebas. El segundo parámetro va a ser el tamaño de la ventana, la ventana es cuántos elementos quiero tomar antes del valor predicho para poder hacer la generación de las nuevas variables. En este caso vamos decir que vamos hacer una ventana de tamaño tres, o sea, tres días antes de la predicción. Vamos a ejecutarlo.

Ok. Ahora, hay que hacer un cambio muy específico cuando estamos trabajando con Pandas. ¿Por qué? Porque si nosotros nada más tomamos la ventana así como tal, y le damos la media y el promedio, lo que está haciendo en realidad es tomar el valor que queremos predecir. Entonces, ya estamos tomando medidas del futuro, algo que sería imposible. Entonces, para ello hay que mover toda la serie de tiempo un día y ahora si tomar el ventaneo. Ok, para mover toda la serie de tiempo

vamos a utilizar la instrucción "shift", más específico en la columna "Close" que sería: df.Close, vamos a hacer un shift, o un deslizamiento de un día y lo vamos a guardar en esta variable "df_shift".

Entonces, si lo imprimimos en pantalla vamos a ver la misma serie de tiempo solamente movida un día. Ahora sí, Pandas nos ofrece una manera cómoda en la cual poder hacer estas ventanas de tiempo y de una vez ahí mismo poder hacer alguna edición de esos valores. Los más comunes son el promedio y la desviación estándar.

Entonces, vamos a utilizarlos. A partir de df_shift, que es nuestra serie de tiempo de deslizada, vamos a utilizar una función "rolling", que sería cómo llaman ellos al ventaneo, y va a ser un ventaneo de tamaño... tamaño de nuestra ventana, que ya definimos que es de tamaño 3. Y una vez que tengamos eso vamos a obtener la media. Y esto no nada más lo va a hacer para la última medida. En realidad lo va a hacer para toda nuestra serie de tiempo. Entonces, lo que vamos a obtener al final va a ser una serie de tiempo con todos los promedios, con los venteneos de tres días anteriores. Ok, lo mismo va a pasar para la desviación estándar, y ahora tenemos seis individuales.

Para cambiarles los títulos, en lugar de que sea un titular de columna, en serie se llama "name" o nombre de la serie. Entonces, lo podemos cambiar directamente, le podemos decir que a esta "roll" que acabamos de crear le cambio el nombre a "mean_roll", es un título que vamos a utilizar provisionalmente, y estándar roll (std_roll), que es la desviación estándar.

Ahora, si nosotros lo visualizamos va a quedar la serie, pero van a estar desfasadas en tiempos. Entonces, hay que reiniciar los índices, que es: los días, para que coincidan todos los valores y podamos ponerlo en un solo data frame. Ok. Vamos a ejecutar todo esto y si lo imprimimos va a quedar precisamente lo que les mencionaba. Si se fijan todas las series empiezan en el primero de junio del 2016 pero obtiene los valores vacíos, ¿por qué? Porque, si obtenemos el ventaneo del primer elemento, no hay datos históricos para poder hacer el ventaneo. Entonces, es imposible realizar el ventaneo. Entonces, queremos eliminar los tres primeros días porque no nos van a servir para poder hacer predicciones. Entonces, tenemos que reajustarlo.

Entonces, lo que vamos a hacer es unir estas series con la serie original y la vamos a cortar los primeros tres elementos. Entonces, primero los unimos, queda de esta forma, si se fijan no puse el desplazamiento original, solamente la media y la desviación estándar. Ustedes ya con más tiempo podrán probar qué pasa si agregamos ese ese deslizamiento de un día o dos días, tres días. Van a encontrar que hay un problema muy particular sobre todo ya cuando entran en el detalle. Pero eso ya lo podemos ver más adelante.

Entonces, quedamos que vamos a eliminar los tres primeros elementos. Para ello podemos usar el indexado de Python que nos dice: a partir del tercer elemento hacia el infinito, o sea, todos nuestro conjunto de datos, guárdalo como df_w. Y listo, ya tenemos nuestros datos listos pero ahora nos falta reservar la porción que vamos a utilizar para pruebas. A diferencia de clasificación y regresión, donde utilizamos el train_test_split, la función específica de Python, lo que vamos a hacer nada más es reservar la última parte de nuestros datos.

Entonces, es una división mucho más sencilla. Le vamos a decir que en test guarde los últimos elementos de nuestro data frame y en train obtengan los primeros menos el tamaño del conjunto de pruebas, ok, que ya establecimos previamente, que van a ser dos meses, o 60 días. Entonces, de prueba solamente vamos a utilizar dos meses.

Ok. Ahora, lo que nos interesa predecir es el valor de cierre. Esta sería nuestra variable dependiente, o "y". Entonces, vamos a hacer eso. Hay que asignarle a "y" el valor de Close para test y para train, y el restante, que sería el mean_roll y el std_roll, lo vamos a guardar como X_test y como X_train. Ok. Entonces, ya tenemos nuestro conjuntos de datos separados listo para poder empezar a hacer predicciones.

Ahora, ¿cómo establecemos las predicciones, o cómo hacemos predicciones? Bueno, ya estuvimos viendo algunos videos de regresión. Podemos tomar cualquiera de esos algoritmos y aplicarlos, y empezar a compararlos cómo funcionan. En esta sección en específico vamos a utilizar una SVR.

Una máquina de soporte vectorial para regresión. Vamos a utilizar de la librería sklearn, de svm…la clase svr, vamos a inicializarla. Vamos a especificarle el gamma solamente para que no nos marque un error o un warning Python en esta versión en específico. Tal vez en su versión ya no sea necesario. Vamos a entrenarla con nuestros datos "x" y "y" de entrenamiento, y a partir de ahí ya podemos hacer predicciones. Podemos decirle que nuestra SVR haga una predicción con el conjunto de datos de entrenamiento. A ese le vamos a actualizar el índice para que coincida, y lo vamos a hacer una serie para que compactado una estructura que podemos estar utilizando. Lo mismo vamos a hacer para la prueba.

Ok. Entonces, ya tenemos todo preparado. Ahora falta graficarlo para poder visualizar qué es lo que está pasando. En primera instancia vamos dejar una figura algo grande, vamos a ponerle un 9x5. Vamos a agregar el conjunto de entrenamiento "y", vamos a agregar el conjunto de pruebas "y" pero en predicción y en valor real. Recordemos que la predicción va con el "_hat", y el real simplemente va así, no lo lleva. Vamos a agregar una leyenda para que imprima estas etiquetas que agregamos, y vamos agregarle un título y lo vamos a imprimir. Ahora sí, nos está generando una gráfica.

Podemos ver todo el conjunto de entrenamiento que es lo azul. En esta parte de aquí, que son los últimos 60 días, ya se ven dos líneas diferentes, se ve una amarilla y una verde. La amarilla es la predicción y la verde es... es los datos reales. Aquí puede ser un poquito difícil de verlo, entonces vamos a tratar de agregarle zoom. Si se fijaron, si fueron observativos, al principio agregamos esta parte que dice "matplotlib notebook", ésta nos permite hacer gráficas de una forma en la que podemos analizarlos dinámicamente.

A veces hay que ejecutarlo dos veces porque se puede presentar un error interno de Python. Es algo que seguramente van a parchar en algunas versiones futuras.

Entonces, regreso a la última celda y la ejecuto. Ahora sí, verán que agrega esta barra gris arriba y estos botones de acá abajo. Lo que nos permite hacer es, por ejemplo, si presionamos este cuadrito nos da la opción de agregar zoom. Entonces podemos visualizar específicamente la parte que nos interesa. Ahora sí, ya podemos ver más claramente las diferencias entre predicción y valores reales. Aunque no es perfecto va más o menos relacionado una con la otra.

Ahora, recuerden que series de tiempo es un problema complicado: hay que ver periocidad, hay que ver ruido, hay que ver análisis del sentimiento, por ejemplo, en internet, noticias, es un problema muy complejo. Entonces, no vayan directamente a... con esto e invertir todo su dinero en la bolsa, verdad. Hay que seguir estudiando al respecto, pero provisionalmente podemos ir midiendo qué tan bien o que tan mal le fue. Para esto se utilizan métricas de error como lo hemos hecho anteriormente. En específico vamos a utilizar el error medio cuadrático es una de tantas métricas de error y les vamos incluso a agregar una lectura recomendada en la cual habla de diferentes tipos de métricas.

No es necesario que las implementen todas. Incluso con una es más que suficiente, pero esta es una que tienen que conocer porque es de las más populares. Entonces, es muy fácil de utilizar, solamente la importamos como sklearn metrics, como error cuadrático medio, la mandamos a llamar donde le pasamos los valores reales y los valores predichos.

Y esto lo igualamos a alguna variable en la cual vayamos a guardar el error. Luego nada más imprimimos y nos da un error cuadrático medio de 0.02. Entonces, esto nos va a decir tal vez si va muy bien no va muy mal, pero no tenemos una referencia clara. Entonces, lo que podemos hacer es poder construir algún otro modelo, medir su error y comparar. De esta forma podemos ir mejorando nuestro modelo para encontrar el que nos va a dar las mejores predicciones.