Resumen del capítulo: Tareas de negocio que involucran al machine learning

¿Qué es el aprendizaje?

El aprendizaje consiste en buscar relaciones. Cuando las relaciones se buscan a través de un algoritmo en lugar de un ser humano, podemos hablar de aprendizaje automático (ML, machine learning).

En los negocios como en la vida, es importante ser capaz de pronosticar la ocurrencia de ciertos eventos o el valor de una cierta variable.

Un variable se llama **objetivo** si puede ser **predicha** o **pronosticada** en función de **características** específicas. Los pronósticos se basan en **observaciones**: ejemplos existentes para los que ya se conocen los valores de característica y variable objetivo.

Para realizar pronósticos exitosos, aprende a **encontrar interrelaciones**. Están basadas en la experiencia y la observación, o **datos empíricos**. La habilidad para encontrar ciertas interrelaciones para pronosticar eventos se llama **aprendizaje**.

Introducción al pronóstico y al aprendizaje automático

Definamos el set de valores de la variable objetivo así:

$$\vec{y}$$

Esto no es una solo número sino un conjunto de varios números. Dichos conjuntos se llaman **vectores** (puede que recuerdes esa pequeña flecha de tus años de colegio). El vector de la variable objetivo es `y` is the vector of the target variable. Cada objeto tiene un **identificador de una observación única** o índice.

Para cada índice `i`, tienes un conjunto de valores de característica o **vector de característica**

$$\vec{x_i} = (x_{i_1}, x_{i_2}, ..., x_{i_N})$$

, donde (`x_i_1, x_i_2, ..., x_i_N`) son los valores de los parámetros N para el objeto en el índice "i".

Los objetos M y los parámetros N se pueden presentar como una tabla o **matriz**. Las filas contendrán observaciones u **objetos**. Las columnas contendrán características. Vamos a definir tal **matriz objeto-característica** como X:

$$X = \begin{pmatrix} x_{1_1} & \cdots & x_{1_N} \\ \vdots & \ddots & \vdots \\ x_{M_1} & \cdots & x_{M_N} \end{pmatrix}$$

Tu tarea es utilizar estos datos para crear una **función** que (relativamente) **refleje la interrelación** fielmente de un fenómeno natural e identifique la relación entre los valores de característica y la variable objetivo (lo llamaremos ŷ: esto es un pronóstico hecho utilizando una función):

$$f(\vec{x}) = f(x_1, x_2, ..., x_N) = \hat{y}$$

Si aplicas esta función a los valores de característica de un objeto particular, obtendrás el **pronóstico de la variable objetivo para esta observación.** Debería acercarse al tamaño de la población observada en el pasado:

$$f(\vec{x_i}) = f(x_{i_1}, x_{i_2}, ..., x_{i_N}) = \hat{y_i}$$

$$\hat{y_i} \sim y_i$$

Cuando tu fórmula produce un **pronóstico** o **estimación** del valor de la variable objetivo que es cercano al valor histórico en la mayoría de casos, básicamente estás encontrando la relación oculta entre los valores. ¿Cómo deducimos esta fórmula? Podemos asumir que se ve así:

$$\hat{y_i} = w_0 + w_1 * w_{i_1} + ... + w_N * w_{i_N}$$

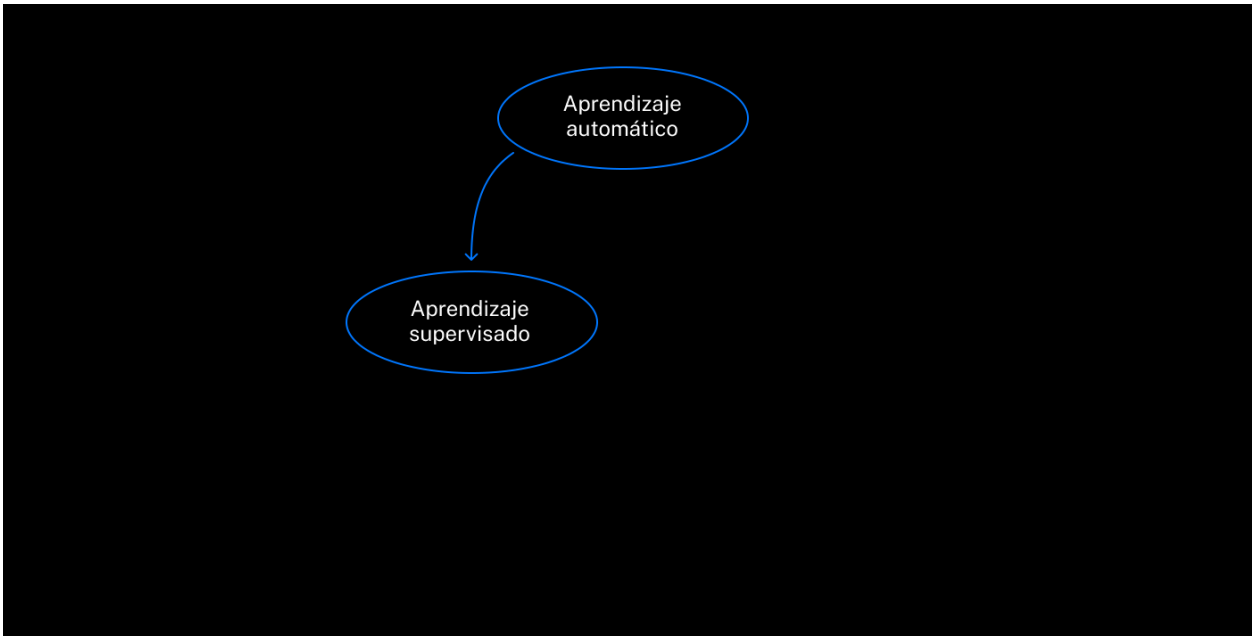
De acuerdo a esta fórmula, el número pronosticado de pingüinos es la suma del coeficiente "nulo" (el número de pingüinos cuando todas las características son iguales a 0) y los valores de las características multiplicados por los coeficientes.

Supongamos que hemos averiguado correctamente la fórmula. Pero, ¿cómo elegimos los valores `w_0, w_1, ..., w_N` (también llamados **parámetros de función** o **pesos**) de tal manera que la ley que estamos deduciendo funcione para todas nuestras observaciones? Siempre puedes delegar la búsqueda de interrelaciones en una máquina (un algoritmo o programa).

El **aprendizaje automático** es **el proceso de búsqueda** de interrelaciones entres los valores en función de varios objetos realizado por una máquina. A medida que el algoritmo procesa enormes matrices de datos, **aprende** de ellos y construye un **modelo mundial** que refleja relaciones ocultas entre procesos u objetos. Además, necesitas ser capaz de distinguir qué tan bueno es un modelo, es decir, con qué precisión sus pronósticos coinciden con la realidad. Discutiremos esto con más detalle en nuestra lección sobre **métricas**. El modelo aprende de un número de observaciones, tal y como lo hacen las personas. Cuantas más observaciones, mejor.

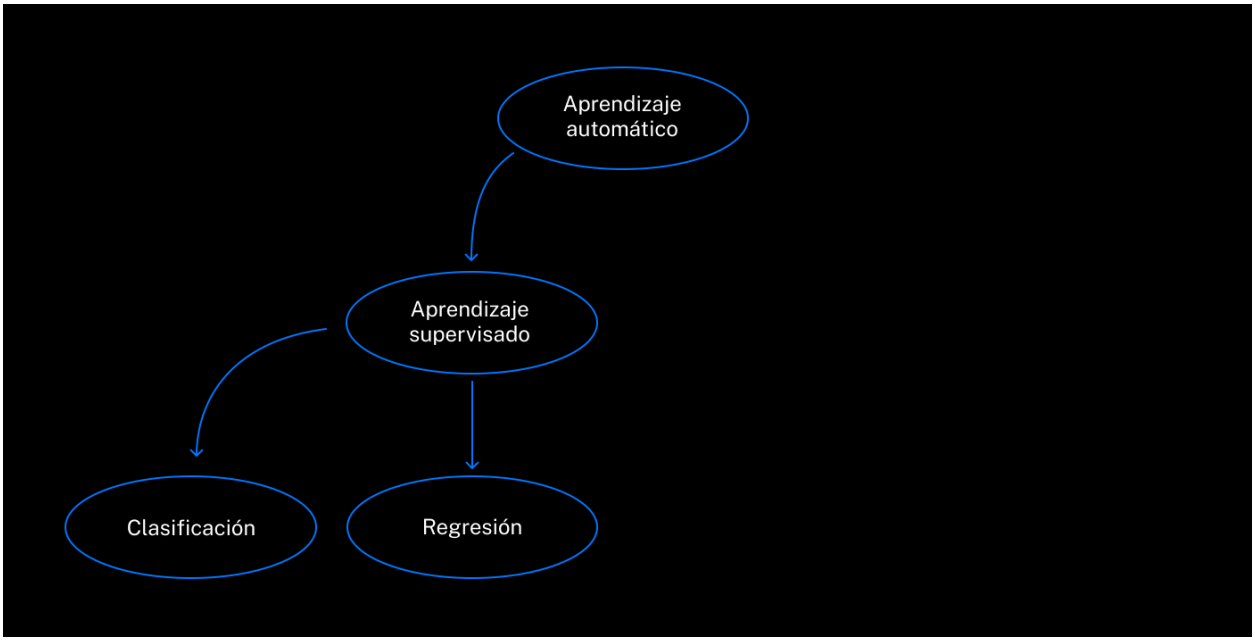
Aprendizaje supervisado

La tarea de la lección previa es un ejemplo de **aprendizaje supervisado**.



En tales casos, a los modelos se les da un gran número de observaciones de entrada, donde cada observación tiene valores de característica (`x`) y el valor de la variable objetivo (`y`) o **label** (etiqueta). Por consiguiente, podrías escuchar a alguien decir: "**Los datos de entrada están etiquetados**". La tarea del modelo es descubrir la relación entre `x` e `y` y aprender a predecir `y` para los objetos nuevos que solo tienen el vector de valores de característica (`x`). En este caso eres tú quien está entrenando el modelo, ya que tú determinas qué puede ser identificado como características (objetos) y qué considerar variable objetivo o **respuesta correcta**.

Los modelos de aprendizaje supervisado se dividen en modelos de **clasificación** y **regresión**.



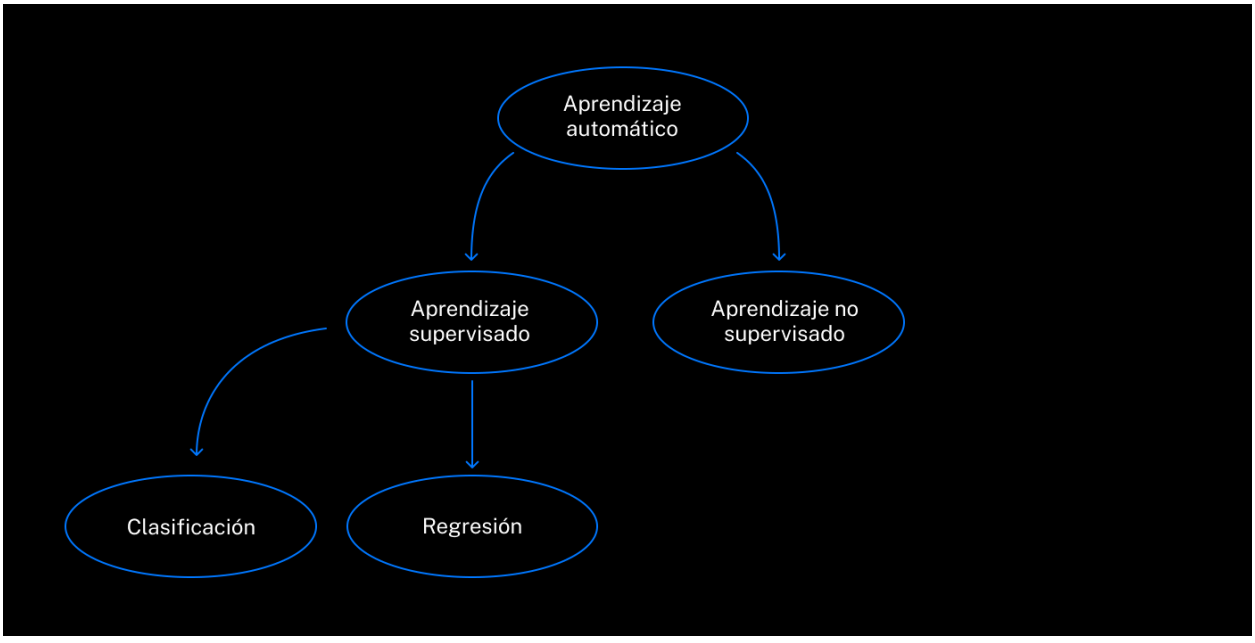
La **clasificación** se utiliza cuando necesitamos conseguir el nombre de la **clase** de cada nuevo objeto como la respuesta del modelo. Puede haber cualquier número de clases (N) pero tiene que ser finito y más de 1. Cuando N=2, la clasificación será **binaria**. Un ejemplo de una tarea de clasificación binaria podría ser la puntuación de crédito. Aquí necesitaremos averiguar si el prestatario devolverá el dinero o no.

Si N>2, es una **clasificación multiclase**. Por ejemplo, si necesitas saber si el cliente pagará el préstamo a tiempo, lo devolverá con retraso o si no lo pagará en absoluto, esto será una tarea de clasificación multiclase.

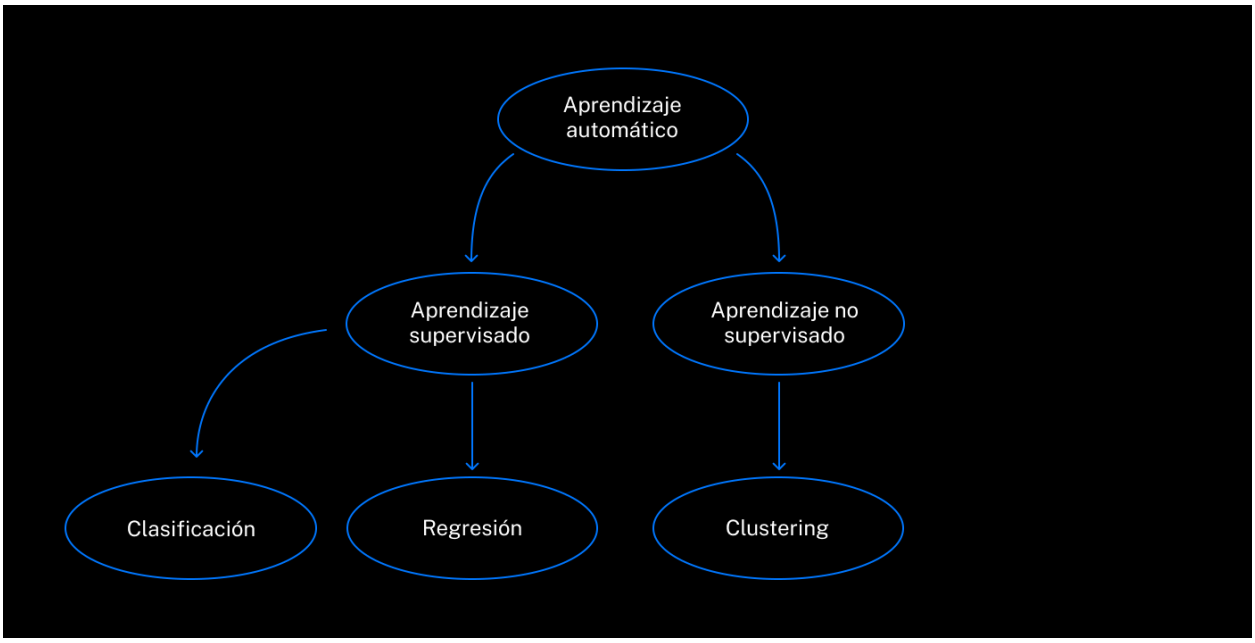
Con **regresión**, la respuesta será una **variable continua**. En los negocios, hay una gran cantidad de tareas de regresión: por ejemplo, pronosticar los beneficios o ingresos de un nuevo punto de venta o estimar los precios de la vivienda en el mercado inmobiliario.

Aprendizaje no supervisado

Has estudiado tareas en las que se conoce la "verdad fundamental" pero también puedes dar datos de un modelo sin proporcionar una variable objetivo. Puedes pasar un gran número de observaciones a un algoritmo sin respuesta (`y`) y entrenarlo para crear interrelaciones entre los propios objetos. El modelo es capaz de identificar qué objetos se parecen entre sí y cuáles no. Al mirar dos manzanas, se puede decir que comparten las mismas características pero una naranja es distinta. Este proceso se llama **aprendizaje no supervisado**. Este es el segundo tipo importante de aprendizaje automático.

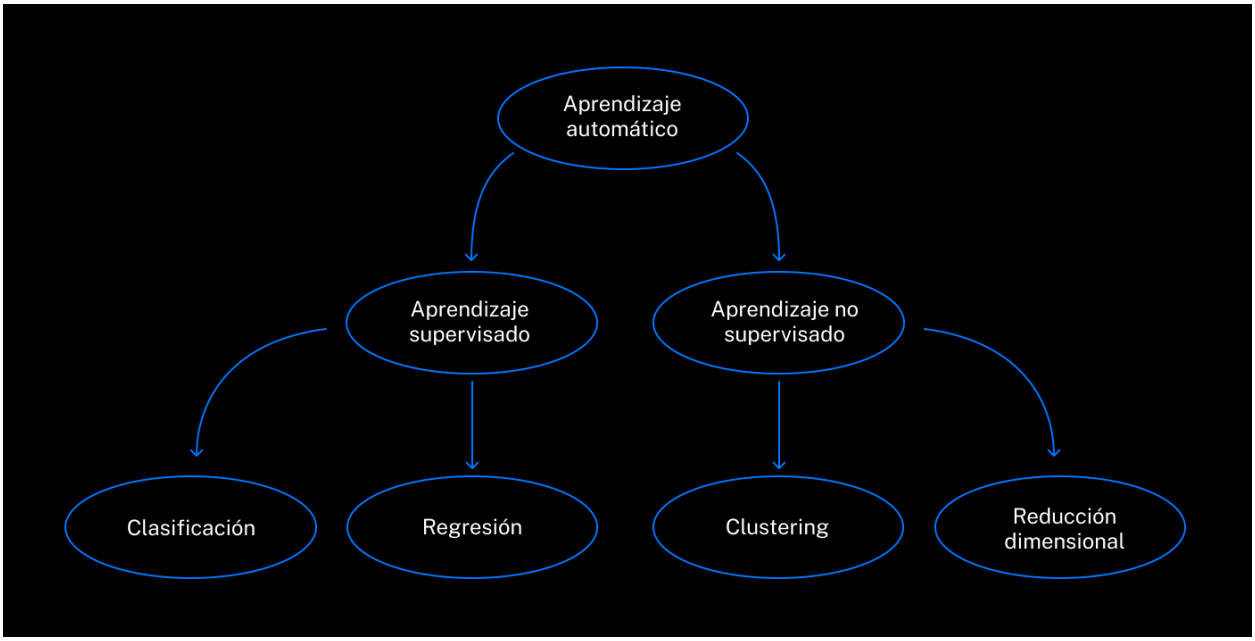


Dividir objetos en grupos se llama **clustering** (agrupamiento). La diferencia principal entre clustering y clasificación es que no hay clases dadas o "respuestas correctas".



Con el clustering, no tenemos clases dadas; el algoritmo es lo que las forma.

El aprendizaje no supervisado también se aplica en **reducción de la dimensionalidad de las matrices de características.**



¿Por qué necesitamos esto? Cuantas más características, mejor, ¿no? Parece ser que no siempre. Una de las razones para la reducción dimensional es que los algoritmos no

funcionan bien cuando hay demasiadas características y no hay suficientes observaciones.

El último tipo importante de ML es el **aprendizaje reforzado**. Aquí el modelo aprende paso a paso y altera ligeramente su algoritmo operativo en cada etapa, utilizando pistas del entorno exterior. El aprendizaje reforzado es un área popular del aprendizaje automático pero vamos a centrarnos en los primeros dos tipos.

Entrenar un modelo en Python: sklearn

Un **modelo** es un sistema de interrelaciones entre características y una variable objetivo o entre observaciones que reflejan la realidad en un alto grado de precisión. Los modelos se entrenan utilizando **observaciones** existentes y se crean con varios métodos o **algoritmos**. Normalmente, "algoritmo" se refiere a un enfoque abstracto de entrenamiento del modelo. Cuando le aplicamos lenguajes informáticos, lo llamamos **implementación de algoritmos**.

En esta sección estudiarás ejemplos de la implementación de algoritmos en Python, el lenguaje con las librerías más convenientes para las tareas de aprendizaje automático. Por ahora solo necesitaremos dos de ellas:

- pandas: para análisis y preprocesamiento de datos
- scikit-learn (sklearn): para la implementación de los algoritmos de aprendizaje automático

Entrenemos nuestro primer modelo en Python con métodos de la librería **scikit-learn**, que generalmente se importa como **sklearn**. La librería sklearn tiene una gran cantidad de herramientas para trabajar con datos y modelos, así que se agrupan en subsecciones. El modulo **tree** contiene el árbol de decisión. En sklearn cada modelo tiene una estructura de datos correspondiente. **DecisionTreeClassifier** es una estructura de datos diseñada para la clasificación con un árbol de decisión. Estudiarás los árboles de decisión más tarde. Importemos la estructura de la librería:

```
from sklearn.tree import DecisionTreeClassifier
```

Luego, crearemos un objeto con esta estructura de datos.

```
model = DecisionTreeClassifier()
```

La variable `model` almacenará nuestro modelo. Cierto, aún no puede hacer predicciones. Para entrenarlo, necesitamos lanzar el algoritmo de entrenamiento.

Los modelos reciben un conjunto de valores de característica, `X`, y una variable objetivo, `y`, como entrada. Normalmente tu DataFrame contiene una columna con los valores objetivo y luego otras columnas. Por ejemplo, digamos que tienes una tabla `data`. Su columna con la variable objetivo se llama `'target'`. Para definir una matriz objeto-característica `X` y un vector variable objetivo `y`, llamaremos al método **drop()** de la librería pandas:

```
y = data['target']
X = data.drop(['target'], axis = 1)
```

El método recibe una lista con los nombres de las columnas a eliminar. El parámetro `axis = 1` indica que es una columna que queremos eliminar.

Ahora podemos crear una interrelación y utilizarla para predecir `y` de la nueva `X`. Para empezar a entrenar, llama al método **fit()** y pásale los datos como un parámetro:

```
model.fit(X, y)
```

Para entrenar el modelo, pásale la matriz con características (`X`) y el vector con los valores de la variable objetivo (`y`).

Para hacer predicciones para un conjunto de datos, solo necesitas llamar al método **predict():**

```
predictions = model.predict(X)
```

Datos de entrenamiento, validación y prueba

Has entrenado tu primer modelo. Pero, ¿cómo sabemos qué tan bien funciona?

Antes de que pases al modelo los datos de entrada reales, necesitas asegurarte de que funciona bien. Puedes entrenar el modelo (utilizando **datos de entrenamiento**) y luego comparar sus predicciones con los valores objetivo reales a partir del mes siguiente. Los datos dados al modelo durante el entrenamiento con el fin de afinarlo se llaman **datos de validación**. Finalmente, probamos el modelo en **datos de prueba** o **de retención**.

Tomemos las 150 000 observaciones y dividámoslas en dos partes desiguales de 100 000 y 50 000. Pasaremos la primera parte al modelo y entrenaremos el algoritmo con ellas. Luego, utilizaremos el modelo para predecir las respuestas para la segunda parte de los datos y comparar los resultados con los valores objetivo reales. Esto nos permitirá afinar el modelo.

Las 100 000 observaciones utilizadas para entrenar el modelo son datos de entrenamiento. Los 50 000 utilizados para probar el ajuste final son datos de validación.

Subajuste y sobreajuste

Cuando entrenamos un modelo y "ajustamos" pesos utilizando datos de entrenamiento, medimos en cada paso qué tan cerca está la respuesta con los pesos elegidos del estado real de las cosas. Por consiguiente, **estimamos el error de entrenamiento** incluso durante la etapa de entrenamiento.

Es casi imposible elegir los pesos apropiados para un modelo lineal o cualquier otro algoritmo para que dé una respuesta correcta para cada observación. El modelo se equivocará con algunas observaciones. A esto se le llama **error de entrenamiento**. Nuestro objetivo es minimizarlo. Si no lo hacemos y el algoritmo que hemos terminado eligiendo es erróneo para la mitad de todas las observaciones, significará que nuestro modelo está **subajustado**. El error que ocurre en tales casos se llama **bias* (sesgo). Una función sesgada (de bias) no tiene en cuenta todas las relaciones dentro de un conjunto dado de datos. Las razones más comunes para el sesgo son:

- El número de muestras o características es demasiado pequeño
- La función es demasiado simple
- Un enfoque incorrecto para seleccionar opciones para la relación objetivo

Tal modelo producirá resultados pobres tanto con los datos de entrenamiento como con los de prueba.

Lo ideal es que un modelo (función, algoritmo) no solo produzca errores rara vez cuando se está entrenando, sino que también funcione bien con los nuevos datos que no "veía" cuando estábamos "ajustando" los pesos y buscando la mejor relación posible. En otras palabras, el modelo debe tener una **capacidad de generalización** alta. Entonces, utilizar el aprendizaje automático será realmente útil.

Imagina un modelo que realiza predicciones realmente precisas sobre los datos de validación. Parece perfecto... pero solo hasta que se dan valores de característica para objetos nuevos. Cuando el modelo demuestra resultados mucho peores en los datos de validación que durante el entrenamiento, se llama **sobreajuste**. Tales errores se denominan **error de varianza**. Esto implica que el modelo está intentando arduamente ajustar los datos y no ignora el ruido: cuando fue entrenado, tuvo en cuenta información excesiva además de las relaciones reales dentro de los datos.

Dividir y validar

Los términos "datos de validación" y "datos de prueba" se utilizan a veces indistintamente y no existe un consenso respecto a sus definiciones. Y nos encontramos lo mismo en las librerías Python: la función que segmenta los datos se llama `train_test_split()`, aunque la podemos utilizar para conseguir datos de validación. No te asustes; en cada caso, toma en cuenta el contexto y deberías ser capaz de averiguar lo que se pretende.

No importa cómo llamemos a los porciones de datos, las preguntas básicas permanecen igual:

- **¿En qué proporción segmentamos los datos?** Normalmente la proporción es 80/20 o 70/30. La norma general es que los datos de entrenamientos deben ser más grandes que los datos de validación pero estos últimos deben ser lo suficientemente grandes para justificar llamar al test "fundamentado".
- **¿Cómo los dividimos?** Vas a aprender dos formas de dividir los datos: **por tiempo** o **aleatoriamente**. El primer método funciona bien cuando las

observaciones previas impactan a las más recientes. El segundo es útil si la estructura de tiempo no es tan importante en el entrenamiento de datos.

Una forma rápida es llamar a la función `train_test_split()` del módulo `model_selection` de la librería sklearn. La sintaxis es la siguiente:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Aquí los datos de entrada para la función son la matriz de característica `X`, el vector de la variable objetivo `y` y el parámetro `test_size`, que controlan la cantidad de datos que serán divididos.

La función devuelve dos matrices de características y dos vectores de variable objetivo, obtenidos al dividir los datos originales de acuerdo con la proporción definida en `test_size`.

Por defecto, `train_test_split()` divide aleatoriamente los datos en la proporción que tú establezcas. Si ejecutas el código varias veces, conseguirás diferentes matrices y vectores.

Para que tu trabajo en este curso sea comparable con el de otros estudiantes, da al parámetro `random_state` un valor de cero cuando dividas los datos:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

El parámetro `random_state` también se presenta en otras funciones y es responsable de la "aleatoriedad.” Por ejemplo, se puede utilizar cuando definimos el algoritmo del modelo:

```
model = RandomForestRegressor(random_state=0)
```

Evaluar la calidad de un modelo

Tras dividir los datos de la mejor forma y en las proporciones correctas, entrénalo con los datos de entrenamiento y comprueba su desempeño con los datos de validación.

Hablaremos de las métricas de los modelos con más detalle en lecciones futuras. Por ahora, miremos a su sintaxis en función del **coeficiente de determinación** o **R-squared**. A esto es a lo que llamamos la parte de la varianza de la variable objetivo que el modelo explica. Esta métrica toma valores entre 0 y 1 y describe cuán precisamente el pronóstico del modelo representa la variable objetivo. Si los pronósticos son perfectos, el valor de R-squared (R2) será 1; si son horribles, será 0. La fórmula del coeficiente de determinación de la muestra se ve así:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \overline{y}_i)^2}$$

Da miedo pero es bastante simple. El numerador aquí es la suma de todos los valores de error cuadrático así que normalmente tus pronósticos son precisos, la fracción será igual a 0, y la métrica será 1. El denominador contiene la suma de las diferencias entre valores y la media. Normaliza tu error para la varianza de la variable objetivo real. Si el error es grande y el valor en sí varía mucho, esto compensará parcialmente la diferencia entre el pronóstico y el hecho.

La sintaxis de R-squared (`r2_score`) es simple: pasa el vector real de la variable objetivo y el del pronóstico como parámetros:

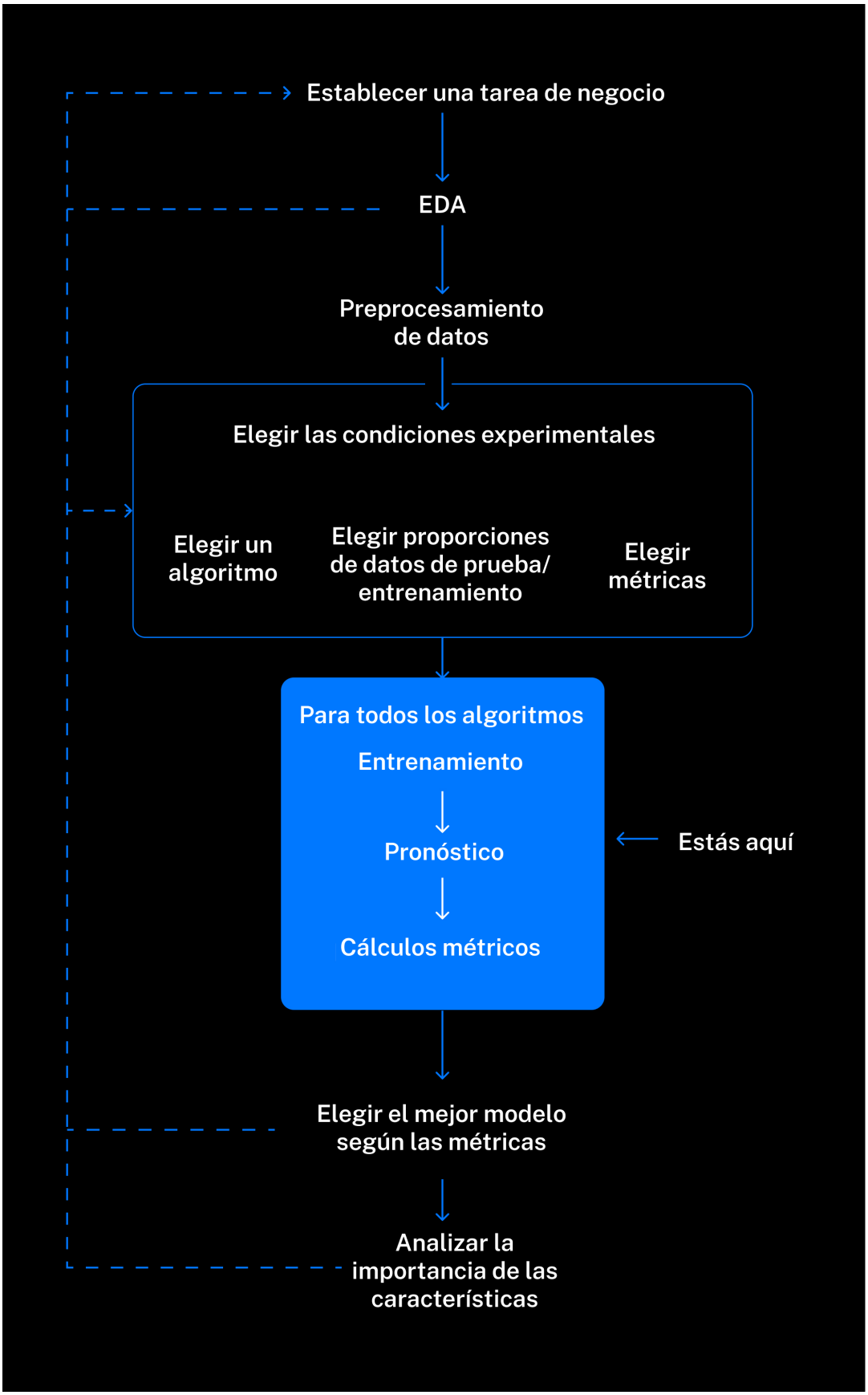
```
metrics.r2_score(y_train, y_pred)
```

La función devuelve un valor: el coeficiente de determinación.

El pipeline de aprendizaje automático

"Aprendizaje automático" se han convertido en las palabras de moda pero el concepto básico es bastante simple. Tenemos un conjunto de observaciones para las cuales se dan valores de característica (vector de carcaterística `X`) y, a veces, un valor de variable objetivo `y`. Dividimos este conjunto en partes: la primera parte se utilizará para entrenar el modelo, la segunda para probarlo y afinarlo, la tercera para realizar pronósticos. Eso es bastante sencillo pero entrenar un modelo es un poco más complicado.

Estas son las etapas básicas del aprendizaje automático:


Establecer una tarea de negocio
EDA
Preprocesamiento de datos
Elegir las condiciones experimentales
Elegir un algoritmo
Elegir proporciones de datos de prueba/ entrenamiento
Elegir métricas
Para todos los algoritmos
Entrenamiento
Pronóstico
Cálculos métricos
Estás aquí
Elegir el mejor modelo según las métricas
Analizar la importancia de las características

La secuencia podría diferir dependiendo de la tarea (aprendizaje supervisado o no supervisado, regresión o clasificación) y el algoritmo. Sin embargo, hay algunos principios generales a los que vale la pena prestar atención.

Definir la tarea

En esta etapa determinas el destino de tu modelo. Traduces una tarea de negocio específica en matemáticas, herramientas analíticas y aprendizaje automático. Asegúrate que entiendes completamente el problema al que el negocio se enfrenta, ya que eso determina la elección del modelo, el algoritmo y las métricas.

EDA

Antes de pasar los datos de entrada a tu modelo para conseguir pronósticos, necesitas llevar a cabo un **análisis exploratorio de datos**. En esta etapa estudias las distribuciones de ciertas características y una variable objetivo, identificas correlaciones entre valores y estudias atentamente el dataset. A veces es útil trazar histogramas de características. Utiliza la función **distplot()** (gráfico de distribución) de la librería seaborn:

```
import seaborn as sns

sns.distplot(df['feature_1'])
```

El análisis exploratorio de datos te permite formular hipótesis iniciales sobre la calidad de los datos y la presencia de anomalías. En este punto ya es posible decir qué características se convertirán en cruciales para el modelo y cuáles pueden, o incluso deben, ser ignoradas. El EDA puede inspirarte para generar nuevas características basadas en las existentes. Tal análisis es importante en tema de negocios y para la calidad de tu modelo actual. Profundizarás en los datos, descubrirás el sentido tras los números y realizarás valiosas hipótesis.

Preprocesamiento de datos

Si solo tienes dos horas para crear un modelo de línea base, probablemente te puedas saltar el preprocesamiento de datos. Pero en realidad, los datos a menudo se **preprocesan**, lo que significa que:

- Nos deshacemos de los valores ausentes.
- Algunas características se transforman (por ejemplo, características categóricas se convierten en cuantitativas).
- Los datos se normalizan o estandarizan (estudiaremos este proceso en el siguiente capítulo).
- Nuevas características se crean en función de las existentes. Este proceso se también se llama **ingeniería de características** y a veces puede mejorar drásticamente la calidad de tu modelo.

Elegir una estrategia de validación

En la lección previa dividiste los datos en conjuntos de entrenamiento y validación y aprendiste por qué es importante estimar métricas con observaciones de retención. En esta etapa deberías elegir cómo crearás el set de validación en función del tipo de datos y la tarea. Hablaremos más de esto en el tercer capítulo.

También es importante en esta etapa asegurarse de que la distribución de valores en los datos de entrenamiento se acerque a la distribución con la que el modelo va a tratar. De otro modo, tu trabajo será en vano.

Elegir un algoritmo

Tienes una caja de herramientas de algoritmos, cada uno con sus propios pros y contras, de la que puedes elegir dependiendo del tipo de la tarea (aprendizaje supervisado o no supervisado) y de la tarea en sí. Algunos algoritmos son más precisos

pero más difíciles de interpretar, otros son más rápidos pero más débiles. Estos son los criterios básicos para elegir un algoritmo:

- Exactitud
- Velocidad
- Interpretabilidad
- Características de los algoritmos individuales: funcionan de forma distinta para las diferentes características

Incluso los algoritmos más simples tienen un número de parámetros que pueden ser ajustados. A veces pueden impactar en la calidad y velocidad del entrenamiento de tu modelo. Normalmente eliges parámetros en las iteraciones: tú entrenas un modelo que tiene ciertos parámetros, estimas las métricas, ves que los resultados son pobres y cambias los parámetros y haces todo el proceso de nuevo. Esto puede continuar de forma infinita pero te enseñaremos a parar a tiempo.

Elegir métricas

Antes de que sigas con el entrenamiento de tu algoritmo, determina cómo evaluarás su desempeño.

Hay un conjunto de métricas estándar para cada tipo de tarea (clasificación, regresión, clustering). Pero es importante no ejecutar simplemente los resultados a través de este conjunto sino entender qué métrica refleja mejor la esencia de un proceso de negocio.

En el segundo capítulo aprenderás qué métricas hay y en qué se diferencian. Quizás crees tu propia métrica algún día para persuadir a tus colegas de equipo de que un modelo es realmente útil.

En esta etapa vale la pena descubrir qué métodos utiliza ya tu compañía para tales tareas. De esta forma serás capaz de comparar la eficacia del aprendizaje automático con una línea de base establecida.

Entrenamiento y pronóstico

Has completado todas las etapas. Luego viene un algoritmo con el tradicional `fit-predict`. En la primera etapa pasaste el conjunto de entrenamiento para que pueda identificar relaciones entre características.

Entonces pasemos a predecir. Tienes datos de retención para los que conoces características y respuestas. En esta etapa tomas las características solas, las asignas al modelo entrenado como entrada y guardas los valores predichos.

Evaluar la calidad de los resultados y elegir el mejor modelo

En esta etapa miras la diferencia entre los valores predichos y reales para los objetos del conjunto de validación. A menudo, los analistas evalúan varios algoritmos y eligen el que tiene las mejores métricas.

Analizar la importancia de las características

Has elegido el algoritmo que es más efectivo en comparación con la línea de base y otros algoritmos, pero aún no está todo listo para lanzar el modelo. Primero necesitas confirmar una vez más que el modelo refleja los patrones e interrelaciones correctos dentro de los datos. ¿Cómo? Por ejemplo, **analizando la importancia de las características**. Esto te permite evaluar no solo las predicciones en sí sino también las razones por las que el modelo las hizo. Por consiguiente, los compañeros y compañeras de otros departamentos serán capaces de confiar en tu modelo y empezar a utilizarlo.

¿Qué sigue?

Encontraste los datos, los transformaste, los dividiste en conjuntos de entrenamiento y validación y elegiste algoritmos potenciales. Entonces vino la etapa `fit-predict`. Elegiste el mejor modelo, interpretaste la importancia de las características y gradualmente te convenciste de que funciona. ¡Magnífico! ¿Podemos introducir ahora el modelo en el flujo de trabajo actual y darle un uso real?

No es tan sencillo: después de pasar por todo el pipeline una vez, probablemente necesites volver a etapas anteriores, hacer cambios y ver cuáles son los efectos. Eso está absolutamente bien. Y no olvides que hay situaciones donde el aprendizaje

automático no encaja, no importa cuán hábil seas tú, ni cuán buenos sean los datos o el modelo.

¿Por qué el aprendizaje automático no es universal?

Sabes más o menos lo que es el aprendizaje automático. Ya es hora de aprender algo igual de importante: cuándo no debería utilizarse. Antes de pasar los datos a un modelo de aprendizaje automático, contesta a estas preguntas clave:

- ¿La muestra es lo suficientemente grande?
- ¿Los datos son de alta calidad?
- ¿Tu modelo es capaz de realizar pronósticos realistas?

Muestras

No hay una respuesta definitiva a qué muestra puede considerarse suficientemente grande; depende de la situación. Los expertos a menudo se guían por las **reglas empíricas**, es decir, las que derivan de la experiencia, cuando deciden sobre el tamaño de la muestra. Tienen en cuenta el número de características, la variedad de valores de la variable objetivo y los detalles de los algoritmos en sí.

De acuerdo con la primera norma, el número de observaciones mínimo requerido en una muestra está linealmente relacionado con el **número de características**. En otras palabras, el tamaño mínimo de una muestra se puede encontrar con la fórmula `s = k * n`, donde `n` es el número de características para cada observación y `k` es una constante que, como muestra la experiencia, a menudo es igual a 10. Si hay 20 características por usuario, entonces necesitarás al menos 200 observaciones para identificar relaciones. Pero si tienes 150 características, ni siquiera un millar de usuarios serán suficientes.

Otra regla empírica es buena para las tareas de agrupación en clústeres (clustering) y toma como punto de partida el **número de clústeres de destino**. Cuantos más clústeres tengas, más difícil será distinguirlos con base en las características disponibles. Si calculas el número de observaciones mínimas requeridas utilizando la

norma previa e incrementas el número de clases objetivo por un factor de `n`, tendrás que incrementar el valor resultante en el mismo grado.

¿Podemos dividir 400 usuarios en 10 clústeres cuando tenemos 234 características? Estás a punto de enfrentarte a la **maldición de la dimensionalidad**. El número de observaciones no será lo suficientemente grande para hacer un agrupamiento para el conjunto entero de características.

La última norma toma como punto de partida la familia a la que pertenece el algoritmo. Por ejemplo, las redes neuronales no funcionan bien con pequeños datasets. Necesitan ser alimentadas con cientos de miles de observaciones.

En resumen: si el número de clientes u observaciones no se mide en miles, no necesitas realmente el aprendizaje automático. De otro modo, ¡juzga cada caso por sus propios méritos!

La calidad de los datos

Imagina que tienes un millón de observaciones. ¡El sueño de un modelo! Pero, ¿la cantidad es lo más importante? ¿Qué ocurre si los datos no contienen realmente la cantidad de información que esperabas tener de ellos para identificar relaciones?

En el aprendizaje automático hay una regla llamada GIGO: basura entra, basura sale (del inglés "garbage in, garbage out"). Si los datos de entrada del modelo son de baja calidad, obtendrás malos resultados incluso si eliges el algoritmo correcto. Puedes encontrarte los siguientes problemas en los datos:

- Ruido
- Valores ausentes
- Errores y valores atípicos
- Cambios en la distribución de los datos a lo largo del tiempo

Trata de averiguar si tu dataset tiene estos tipos de problemas en la etapa EDA. Puedes resolver algunos de ellos mediante el preprocesamiento.

Pero podría haber casos en los que no puedes afectar a la calidad de los datos. Por ejemplo, si para una cierta característica el porcentaje de valores ausentes es más del 50% y las observaciones no tienen una correlación temporal (es decir, no serás capaz

de "estirar" el valor de períodos previos), tendrás probablemente que deshacerte de la variable.

La **variabilidad en los datos** es otro problema que puedes encontrarte. Las distribuciones de las características en los datos de entrenamiento, validación y prueba deben ser similares o el modelo será inútil.

Modelos de baja calidad

Esto también ocurre. Parece ser que tienes una gran cantidad de datos de alta calidad pero las métricas podrían decirte que no puedes predecir la variable objetivo elegida utilizando los datos que tienes. ¿Por qué? Porque tus características no están relacionadas con tu variable objetivo así que no puedes pronosticar el valor.

Los datos sobre el número de visitantes a cadenas de restaurantes estadounidenses no te ayudarán a pronosticar la población de pingüinos africanos, sin importar cuán precisos sean los datos. Probablemente obtengas un error relativo cercano al 100% y un valor de R-squared cercano a 0.

O a lo mejor los datos son demasiado ruidosos y volátiles. Es casi imposible encontrar una "buena señal" en tal conjunto. En casos como estos, el aprendizaje automático no funcionará; necesitarás utilizar otros métodos de modelado matemático.