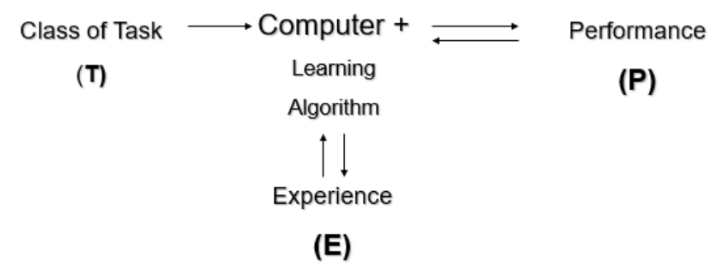
**Introducción a Machine Learning I**

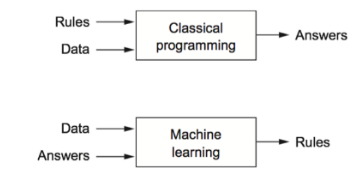
**Introducción:**

**Machine Learning**: Campo de estudio que da a las computadoras la capacidad de aprender sin estar programado explícitamente / Forma de extraer conocimiento a partir de los datos / Se dice que un programa de computadora aprende de la Experiencia **E** con respecto a alguna Tarea **T** y la medida de rendimiento **P**, si su desempeño en la tarea **T**, medido por **P**, mejora con la Experiencia **E**.



Ejemplo: Tenemos una tarea **T** que es reconocer y clasificar palabras escritas a mano; la medida **P** es la cantidad de palabras correctamente clasificadas y la Experiencia **E** es una base de datos con las palabras escritas a mano y su clasificación.

**Programación Tradicional vs Machine Learning:**

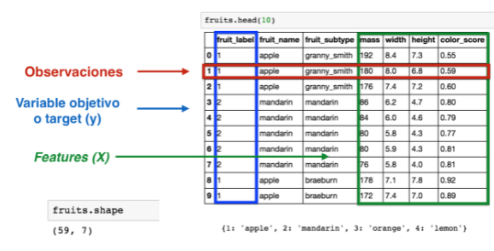


En la **programación tradicional**, los sistemas **son programados** con las **reglas o instrucciones** necesarias para dar **respuestas** en función de los **datos ingresados**. Ejemplo: Sistemas de cálculo, que debe contener las reglas de validación, consistencia y cálculo de los datos ingresados para emitir una respuesta.

En **Machine Learning** le damos a los modelos o programas los **datos de entrada** y sus **respuestas asociadas.** Se puede considerar que **el algoritmo** **aprende de los datos**, **generando las reglas** que luego aplicaremos a **nuevos datos de entrada** para **predecir la respuesta**.

**Diseño de los Sistemas de Aprendizaje:**

Una vez que tenemos un Dataset validado y analizado, veamos sus partes:



**Observaciones** (filas): Objetos individuales descriptos por el Dataset.

**Features** (Columnas): Características de las muestras. Por convención, el conjunto o la matriz de features suele referenciarse con una **variable** llamada **X**. Se asume que la matriz de features tiene una forma [n\_samples, m\_features] y frecuentemente se almacena como un array de Numpy o un DataFrame de Pandas.

Muchas veces, una de las features del Dataset tiene un significado especial: representa un valor que clasifica a cada observación. Por convención se la denomina como **y**; es la **variable target** (**objetivo**), porque puede predecirse a partir de las otras features.

**Tipos de Aprendizaje**: Hay dos tipos de aprendizaje automático:

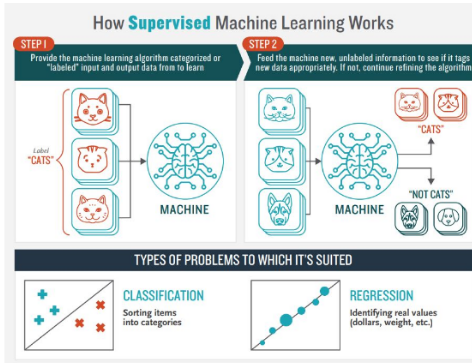
1. **Aprendizaje Supervisado.**
2. **Aprendizaje No Supervisado**

La diferencia entre ambos aprendizajes es **si hay o no** una **variable objetivo a predecir**.

Los dos métodos **deben ser testeados y validados en su performance**. En el caso de los **métodos supervisados**, hay que darle tanto datos de ***input*** como de ***output* asociado** para generar las reglas; mientras que en los **métodos no supervisados**, solo hay que dar datos de ***input****.*

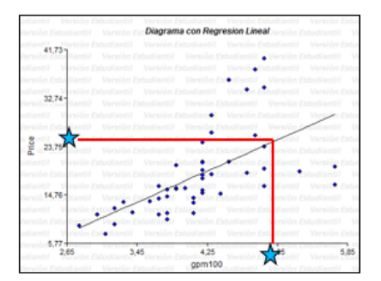
**Aprendizaje Supervisado**

Los algoritmos de aprendizaje supervisado usan las **features** y la **variable target asociada** para crear un modelo predictivo. Al recibir una nueva observación, el **modelo predice el valor del target asociado**.



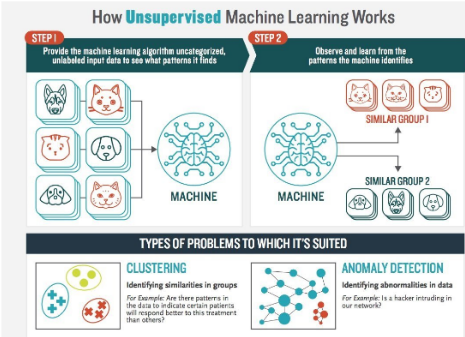
La **Supervisión del aprendizaje** se da cuando le damos datos al modelo para los cuales **ya conocemos de antemano qué resultados deben dar**. Entonces tenemos el **valor real** y el **valor predicho** y podemos compararlos. Según qué tipo de problema estemos abordando, hay diferentes **métricas de performance** que podemos computar para saber qué tan acertadas fueron las predicciones. Si tenemos una **variable objetivo que es una variable continua o cuantitativa**, entonces se trata de un **modelo de regresión**. En cambio, si nuestra **variable objetivo** es una **variable categórica o cualitativa,** entonces estamos ante un **caso de clasificación.**

Ejemplo de aprendizaje supervisado: Tenemos un dataset de autos, con 2 features: el consumo de combustible y el precio del auto. Con un **algoritmo de aprendizaje supervisado**, que tome como feature el consumo de combustible y como variable objetivo el precio, genera una recta que se puede usar para predecir a partir de un consumo determinado, qué precio podríamos esperar:



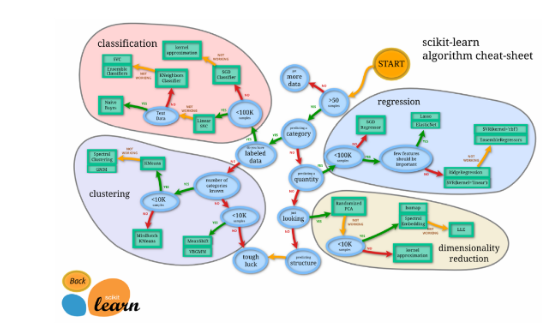
**Aprendizaje No Supervisado**

El aprendizaje No Supervisado se caracteriza **por prescindir de una variable target**. El mismo **algoritmo genera los labels o targets** para cada observación. Su foco se encuentra en **descubrir patroneso estructuras subyacentes en la información**; no evidentes a priori.





**Scikit Learn:** Es una biblioteca de Python que provee implementaciones eficientes de un gran número de los algoritmos más usados mediante una interfaz estándar y consistente; tanto para aprendizaje supervisado como para no supervisado.



Se basa en otros paquetes, tales como Numpy, Pandas y Matplotlib. Permite hacer análisis de textos también. Un equipo de expertos elige, agrupa e implementa los modelos. Scikit-learn incluye la mayoría de las tareas de *machine learning* con una interfaz consistente.

**Metodología:** Existen distintas metodologías para desarrollar modelos de *Machine Learning*. Es importante entender los conceptos detrás de cada paso; la implementación puede variar en función de la metodología adoptada y el tipo de algoritmo a utilizar.

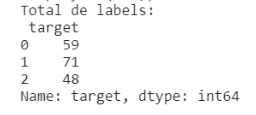
1. **Seleccionar** una **Clase de Modelo.**
2. Elegir los **Hiperparámetros del Modelo**.
3. **Preparar los datos** en una Matriz de **features** y un vector **target**.
4. **Separar** los **tests de entrenamiento** y **testing**.
5. **Ajustar el modelo** a los **Datos de Entrenamiento.**
6. **Predecir** **etiquetas** para **datos desconocidos.**
7. **Evaluar** la **performance** del modelo.

Ejemplo práctico: Se toma un dataset de vinos que provee scikit-learn. **X** es nuestro conjunto de features de 178 vinos; mientras que **y** es la **variable target**, con tres etiquetas distintas.

**En Python**:

(X, y) = datasets.load\_wine(return\_X\_y=True, as\_frame=True)

print(‘Total de labels\n', y.groupby(y).count())



La variable target tiene sólo 3 valores posibles (0, 1 o 2). Estas son las categorías que queremos predecir. Con este dataset intentaremos construir un modelo de aprendizaje supervisado, que clasifique cada instancia(registro, observación) en una de las tres categorías.

**Paso 1: Selección de una Clase del Modelo**. Queremos resolver un problema de **Clasificación**. Entre los modelos de clasificación disponibles, tenemos que seleccionar cuál vamos a usar. Scikit-learn está estructurada en módulos, de forma tal que cada clase de modelo esté en un módulo específico que engloba funciones de la misma familia. El dataset con que contamos y el tipo de método que queremos aplicar sobre los datos nos ayuda a seleccionar la clase de modelo a utilizar. Para este caso, vamos a usar un **árbol de clasificación**. Dicho modelo se encuentra en el **módulo Tree**. La clase se llama **DecisionTreeClassifier**. Otros módulos disponibles para problemas de clasificación además de Tree son **SVM** y **Naive Bayes**. **Este no es un paso menor.** Una vez analizado exhaustivamente el Dataset, podemos saber cuál podemos aplicar y mediante la comparación de la performance de distintos algoritmos podemos decidir cuál es el más adecuado para nuestro problema.

**En Python: from** sklearn.tree **import** DecisionTreeClassifier

**Paso 2: Elección de los Hiperparámetros del Modelo:**

Un **hiperparámetro** es un **parámetro** cuyo valor se utiliza para **controlar el proceso de aprendizaje**. En cambio, los valores de los parámetros derivan a través del entrenamiento.

Los Hiperparámetros pueden clasificarse como:

* **Hiperparámetros de Modelo:** No se pueden inferir al ajustar al conjunto de entrenamiento porque **se refieren a la tarea de selección del modelo**. Son los parámetros del **constructor del modelo**.
* **Hiperparámetros de Algoritmo: En principio no influyen en el rendimiento del modelo**, aunque **afectan la velocidad y calidad** del **proceso de aprendizaje**. Son los **aprendidos durante el entrenamiento**.

**Diferentes algoritmos** de aprendizaje **requieren diferentes hiperparámetros**. Hay **algunos** algoritmos **simples** que **no requieren ninguno** (IE: regresión de mínimos cuadrados ordinarios).

**Al tener** los **valores** de los **hiperparámetros**, **el algoritmo aprende** a partir de los datos (en entrenamiento) **los valores de los parámetros**. Con **distintos hiperparámetros** podemos crear **distintas instancias de la misma clase** del modelo seleccionado. De forma tal que una **misma clase de modelo** con **diferentes configuraciones de hiperparámetros** puede dar **resultados muy distintos**.

Hay que **establecer los valores de los hiperparámetros antes** de que el **modelo sea ajustado a los datos**.

Para **DecissionTreeClassifier** vamos a darle al hiperparámetro **max\_depth** un valor de 2, indicando que el modelo entrenado tendrá como máxima profundidad 2.

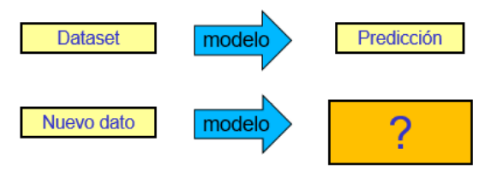
**En Python:** tree\_instance = DecissionTreeClassifier(max\_depth=2)

Una vez creada esta instancia, se puede volver en cualquier momento, cambiar los valores de los hiperparámetros y volver a entrenar el modelo. Consultando la documentación de cada clase podemos ver qué hiperparámetros podemos establecer.

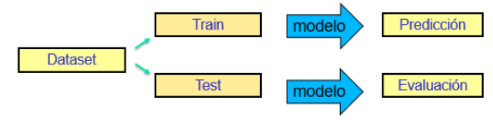
**Paso 3:** **Preparar los datos de una matriz de *features* y un vector *target*:** La matríz de features **X** puede ser un **array bidimensional de Numpy** o bien un **DataFrame de Pandas**. El vector objetivo **y** puede ser un **array de Numpy** o bien una **Series de Pandas**.

En modelos de aprendizaje no supervisado este paso no es necesario.

**Paso 4: Separar los sets de Entrenamiento y Testing:** El objetivo con *machine Learning* es **generalizar** a partir de lo aprendido para poder predecir la variable target de observaciones nuevas para el modelo. Para lograr esto, el modelo necesita aprender a partir de los datos que tenemos. Y para saber si funciona bien, podemos comparar sus predicciones contra los valores reales. Entonces, si queremos saber cómo se comportará el modelo ante datos que nunca vio, lo que podemos hacer es dividir nuestros datos en dos conjuntos, de entrenamiento y testing.



Se dividirán los datos de **X** en un *training set* y en un *testing set*. El *training set* se usará para que el modelo aprenda. Una vez hecho el aprendizaje, se usa el modelo en los datos del *testing set* y se comparan los valores predichos de **y** contra los valores reales:



Podríamos hacer la división en estos dos grupos a mano, pero con Scikit-Learn, contamos con la función **train\_test\_split()** que podemos importar de su módulo **model\_selection**. Este módulo tiene herramientas para la selección y evaluación de modelos.

**En Python:** **from** sklearn.model\_selection **import** train\_test\_split

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, random\_state = 1)

El resultado de esta función serán dos matrices de *features* y dos vectores target, correspondientes a los conjuntos de entrenamiento y testing.

**Paso 5: Ajustar el Modelo a los Datos de Entrenamiento:** Con el método **fit()**, el modelo aprende a partir de los datos de entrenamiento.

**En Python:** tree\_instance.fit(Xtrain, ytrain)

Por convención en scikit-learn, todos los atributos que representan los **parámetros** que el modelo aprendió durante el entrenamiento se identifican con “\_” después de sus nombres. En el caso de los árboles de decisión, uno de estos atributos posibles es **tree\_**.

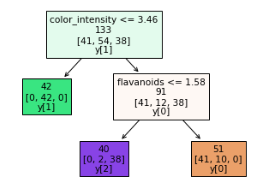
Si queremos conocer el número de nodos del árbol que arman el modelo entrenado, podemos usar **en Python:** tree\_instance.tree\_.node\_count

En este ejemplo da 5. Se puede graficar, **en Python:**

**from** sklearn.tree **import** DecissionTreeClassifier, plot\_tree

plot\_tree (tree\_instance, feature\_names=Xtrain.columns, filled=True, class\_names=True, label=None, impurity=False)

plt.show()



El primer nodo usa la feature color intensity. Si es <= 3.46 entonces lo clasifica con *label1*; caso contrario para a estudiar la feature flavanoids. Si es <= 1.58, entonces lo clasifica con *label2****;*** caso contrario, con *label0*.

**Paso 6: Predecir Etiquetas para Datos Desconocidos:**

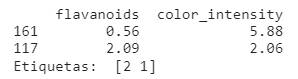
Tenemos al modelo entrenado. Ahora queremos evaluar su performance al alimentarle datos nuevos para él, pero con resultado esperado conocido para nosotros. Para esto, usamos los datos del *test set*, usando el modelo ya entrenado para predecir la variable target. En el paso 7, compararemos estas predicciones con el verdadero valor de cada una de las observaciones de *test set*. Scikit-learn tiene el método **predict()** para asignar a una variable las predicciones del modelo en todas las observaciones del *test set*.

**En Python:** ypred = tree\_instance.predict(Xtest)

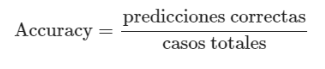
Para ver algunos resultados a ver cómo funcionó, sabiendo que los features que entran en juego son color\_intensity y flavanoids:

print(Xtest.iloc[0:2,[6,9]])

print(‘Etiquetas: ’,ypred[0:2])

****

**Paso 7: Evaluar la Performance del Modelo:** Para evaluar el desempeño debemos **comparar** las **predicciones del modelo** contra los **valores reales**. En el ejemplo en estudio, podemos usar la métrica **accuracy.** Esta puede computarse con el método **accuracy\_score()**. Lo que da es la proporción de etiquetas predichas que coinciden con el valor real:



**En Python: from** sklearn.metrics **import** accuracy\_score

accuracy\_score(ytest, ypred)



Nos dio un accuracy de 91,1%. 91 de cada 100 casos los está clasificando correctamente!

Scikit-learn cuenta con un **módulo de métricas** con funciones específicas para evaluar la performance de un modelo.

**Problemas de Ajuste:** El único propósito de los modelos de Machine Learning es **generalizar bien**.

**Generalización:** Término que indica en qué medida se aplica lo aprendido por el modelo a datos nuevos para obtener una predicción correcta. Un modelo no podrá identificar bien perros si el único dato que le damos es una imagen, o muchas imágenes de una misma raza.

Un **modelo demasiado simple** no podrá aprender lo suficiente a partir de los datos de entrenamiento; por ende, los resultados cuando lo probemos en testing serán malos **(underfitting o subajuste)**.



Un **modelo excesivamente complejo** se amoldará tanto a los datos de entrenamiento que no podrá generalizar a datos desconocidos **(overfitting o sobreajuste)**:



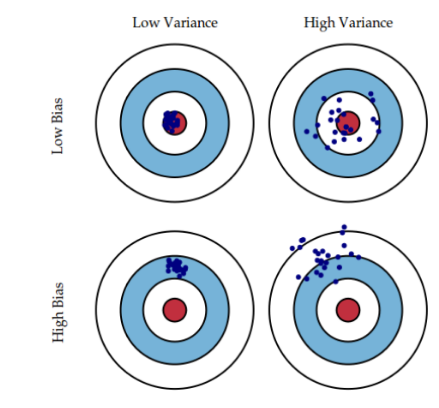
La situación deseable está en un punto intermedio:



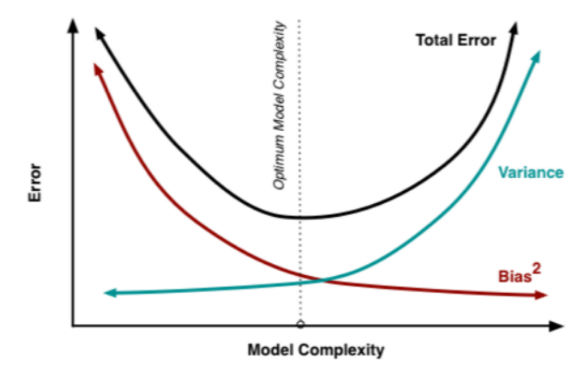
**Sesgo – Varianza:**

El **Error de Sesgo** es un error de suposiciones erróneas en el algoritmo de aprendizaje. Con un sesgo alto, el algoritmo puede perder las relaciones relevantes entre las features y los resultados. Esto es un caso de **underfitting.**

La **Varianza** es un error de sensibilidad a pequeñas fluctuaciones en el conjunto de entrenamiento. Una alta varianza puede hacer que un algoritmo modele el ruido aleatorio de los datos de entrenamiento. Esto es un caso de **overfitting**.

****

Hay que tratar de minimizar estas dos fuentes de errores para que los algoritmos de aprendizaje supervisado puedan generalizar más allá de su conjunto de entrenamiento:



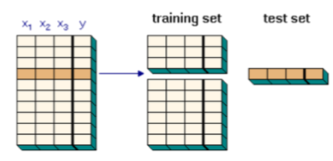
Por lo tanto, **underditting** se relaciona con modelos que tienen alto sesgo y baja varianza, mientras que **overfitting** se relaciona con modelos que tienen bajo sesgo, pero alta varianza.





**Cross Validation**: Consiste en armar multiples (k) sets de entrenamiento y validación a partir de un mismo conjunto de datos. Luego, se realizarán k entrenamientos y validaciones independientes sobre distintos conjuntos; todos extraídos a partir del conjunto de datos original. A este procedimiento se lo llama **cross-validation** (**validación cruzada**)





Se divide el data set en k grupos, de los cuales uno se reserva para test y el resto para training. En cada iteración se va cambiando el grupo que se usa para test. Entonces, prueba a prueba, **los datos de entrenamiento no son idénticos y los datos de validación nunca se repiten.** Por esto se llama validación cruzada. Al hacer múltiples ***folds*** se le da más robustez estadística al análisis, porque promediamos varios resultados con distintos sets de entrenamiento y validación. Normalmente se usan 3, 5 o 10 folds.