# Capítulo 16
# ENTRENAMIENTO Y EVALUACIÓN DE MODELOS

En este capítulo vamos a entrenar y evaluar modelos a partir de conjuntos de datos y uno o más algoritmos de aprendizaje automático. No entraremos en redes neuronales artificiales, contenido que veremos en el siguiente capítulo, si no que nos centraremos en unos pocos algoritmos de base analítica o estocástica.

Antes de empezar es importante recordar que algoritmos de aprendizaje automático para crear modelos hay muchos; y cada uno de ellos posee su propia estrategia para aprender a partir de los datos. Sólo tienen una cosa en común, las matemáticas.

El resto del capítulo está dividido según el algoritmo usado, empezando por uno sencillo (regresión lineal) y aumentado la complejidad. Para cada modelo creado se hará una evaluación y allí en donde sea posible se compararán los modelos.

## 16.1 Regresión lineal

| YearsExperience | Salary |
|---|---|
| 1.2 | 39344.0 |
| 1.4 | 46206.0 |
| 1.6 | 37732.0 |
| 2.1 | 43526.0 |
| 2.3 | 39892.0 |

La regresión lineal es el enfoque de aprendizaje automático que tiene como fin modelar la relación entre una variable dependiente (etiqueta) y una o más variables independientes (características). El caso más elemental es la regresión lineal simple, donde se tiene una sola variable independiente.

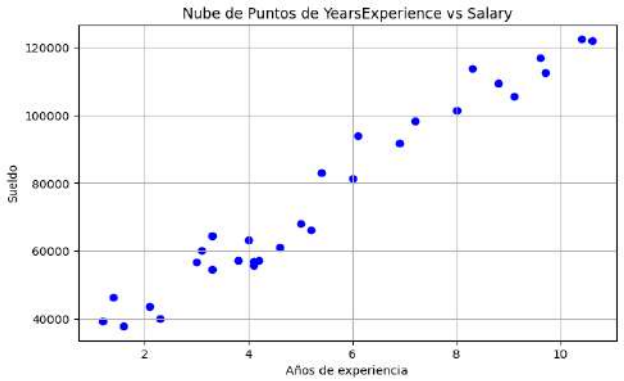
Para empezar, vamos a trabajar en este ejemplo con un *dataset* muy simple, y pequeño (ver a la izquierda); uno con 30 muestras, cada una de ellas con 2 columnas: *YearsExperience* y *Salary*. Este conjunto de datos relaciona los años de experiencia del trabajador con su salario. Bien, ya tenemos los datos. Vamos a **formular el problema**: queremos crear un modelo que, a partir de la experiencia en años de una persona en un puesto de trabajo, prediga su salario.

Evidentemente vamos a cometer un error. Sólo con los años de experiencia no podemos adivinar el salario de la persona, pero podemos estimarla respecto a lo que los datos nos enseñan. Y ahí está el aprendizaje.

Los datos nos dan una muestra de la realidad y nosotros vamos a crear un modelo muy simple que va a generalizar esa realidad. Al aplicar al modelo nuevos datos (años de experiencia), obtendremos el salario que debería tener la persona, según los datos originales.

En el caso de la **regresión lineal simple**, lo que pretendemos es crear un modelo con esta forma: $Ax+B=y$

Este modelo tiene dos incógnitas ($x$ e $y$, respectivamente años y salario), y dos parámetros, $A$ y $B$. Una imagen puede ayudar a esclarecer lo que acabamos de decir.




La imagen anterior muestra la gráfica de nuestro *dataset*, aquel que relaciona años de experiencia y sueldo. Nuestras vivencias nos dicen que cuantos más años pase una persona en una empresa y más experiencia acumule, más elevado será su sueldo.

Esta gráfica está formada por 30 puntos (pares años/sueldo); y representados en forma de nube nos da una idea basada en datos de ese hecho. En el eje de las $x$ (abscisas) tenemos los años de experiencia; fíjate en el valor 6 años de experiencia, sube por el eje $y$ (ordenadas) y obtendrás lo que los datos dicen del salario de una persona a los 6 años de experiencia: poco más de 80.000. Pero ¿qué salario puede esperar una persona con 10 años de experiencia?

Precisamente por eso necesitamos una recta $Ax+B=y$ donde al introducir un año ($x$) nos devuelve el salario ($y$). Sólo necesitamos calcular los términos A y B; y hacerlo de tal forma que sean los mejores términos posibles. ¿Y cuales son los mejores términos posibles? Los que minimicen el error que cometes al usar la línea como modelo.

Veamos un código que los calcula. Partiendo de un *dataset* ya cargado (en el código llamado *data*) calculo $A$ y $B$ usando una función de la librería *numpy* (con *np* como alias). Esta función haya una pareja de $A$ y $B$ tal que minimice la media del cuadrado de los errores:

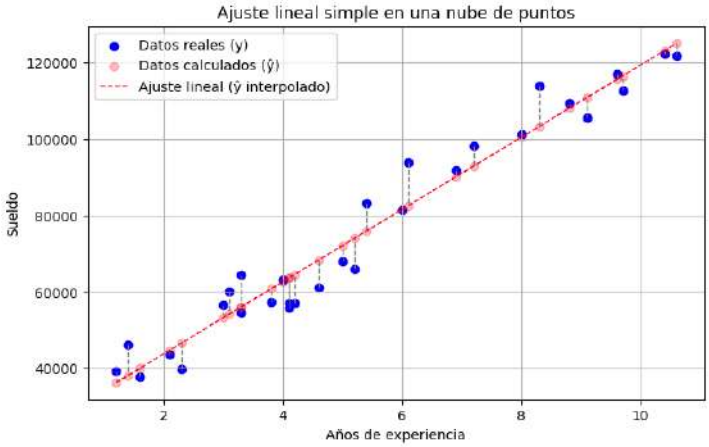
```python
import numpy as np

# Realizar la regresión lineal
X = data['YearsExperience']
y = data['Salary']

# Calculo los términos de la recta
A, B = np.polyfit(X, y, 1)

ŷ = A * X + B # ŷ son los datos predichos
```

Después de la capacidad de crear modelos, lo más importante es poder evaluarlos. Poder conocer cuanto[230] fallamos si tomamos esos modelos como referencia.




Representemos lo que acabamos de decir. La imagen de la izquierda es la misma que la anterior, pero ahora muestra la línea calculada y los errores que cometemos al tomar este modelo como referencia. Los errores son las líneas verticales que van desde el punto (del *dataset*) hasta su proyección vertical en la línea (el modelo).

Imagina que sumamos todos esos errores (líneas verticales) y hayamos su media. Realmente tenemos que sumar su valor absoluto, ya que habrá errores positivos y negativos. Esta métrica recibe el nombre de *Median Absolute Error*, más conocida como MAE:

$$MAE(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y-\hat{y}|$$

Pero también podemos usar la métrica MSE (*Median Squared Error*):

230 *Observa el gráfico, los puntos y la recta, pero sobre todo imagina los infinitos puntos que no están; ¿qué patrón seguirían esos puntos? Muy probablemente el mismo que los puntos que sí están. Si conseguimos un dataset que represente los datos que no están (y que deseamos predecir) podemos crear un modelo predictivo muy bueno. Al menos tan bueno como la representatividad que tiene el dataset que usemos.*

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$$

En nuestro *dataset*, el MAE es de 4644 y el RMSE[231] 5592. No esta mal. Aunque hemos hecho trampas. No hemos separado el conjunto de datos en *trainset* y *testset*, como indicamos en §14.4 que debíamos hacer. Ya verás porqué.

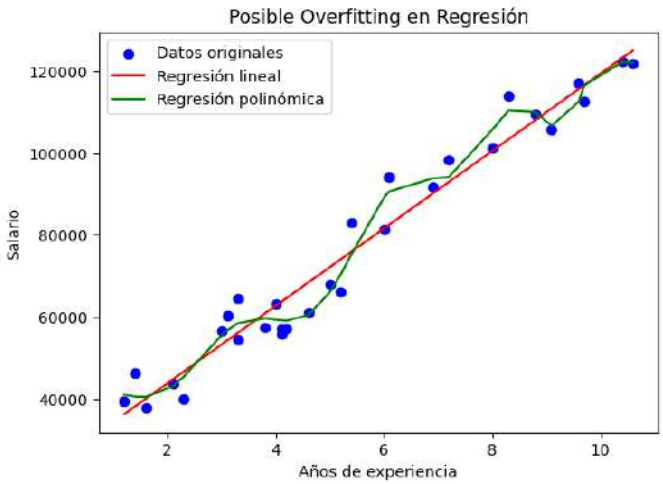
La siguiente figura muestra de nuevo la nube de puntos y la línea (roja) del modelo de regresión lineal. Pero podemos ver un nuevo elemento, la curva que representa a un modelo lineal polinómico. Este modelo intenta buscar una curva que se aproxime mas a los puntos mediante la formula:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_n x^n$$

En donde el aprendizaje consiste en estimar los términos $\beta$ del polinomio de grado $n$. En nuestro caso hemos empleado n=15 para calcular el polinomio, con el siguiente código:

```
1  poly = PolynomialFeatures(degree=15)
```

Un modelo de regresión polinómica.




Volviendo a la figura, **¿qué modelo se ajusta mejor a los datos del *dataset*?** El modelo polinómico. Pero **¿qué modelo predice mejor a partir de los datos del *dataset*?**

"Ajustarse mejor" y "predecir mejor", son conceptos diferentes, aunque relacionados, y como lo que queremos es predecir, nuestro objetivo es crear modelos que predigan mejor a partir de los datos. Esto es, modelos que ajusten mejor, no a los datos del *dataset*, sino a los datos que aún no tenemos. Por eso la respuesta a ¿qué modelo predice mejor? Es una pregunta trampa, pero que de-

231 *Raíz cuadrada del MSE.*

bemos afrontar. De hecho ya sabemos cómo hacerlo.

Hoy en día es difícil viajar al futuro para traer nuevos datos que, para nosotros en este momento, aún no tenemos y que serían muy útiles para probar un modelo que hemos creado. Pero podemos "simular" el resultado de este viaje temporal, ¿cómo? dividiendo el *dataset* en dos partes, el *trainset* y el *testset*; para, a continuación, entrenar el modelo con el *trainset* y probar que tal funciona con el *testset*. En nuestro caso tenemos 30 muestras, podríamos tomar 24 como *trainset* [80%] y 6 como *testset* [20%]. Al hacerlo tenemos la certeza de que hemos probado el modelo con datos que no han sido usados durante el entrenamiento y, si los resultados de la evaluación son buenos y con la información disponible hasta el momento, podemos afirmar que es un buen modelo. Cuando un modelo se ajuste muy bien a los datos de entrenamiento, pero prediga mal, decimos que este modelo está haciendo ***overfitting*** [sobreentrenando]. Volveremos a esto más tarde.

## 16.2 Regresión logística

La regresión logística es un tipo de análisis de regresión que se utiliza para predecir el resultado de una variable categórica en función de las variables independientes. Es la versión del modelo de regresión que acabamos de ver, pero para predecir un conjunto de valores limitados llamados clases. Es, por tanto, una clasificación.

Vamos a retomar el *dataset* que vimos en el apartado 14.3.1 [Máquinas de Soporte Vectorial], aquel que nos entregaba 569 muestras provenientes de biopsias, cada una de ellas con 30 variables, para determinar si un cáncer era maligno o no. Queremos crear el modelo de regresión logística que prediga 1 [benigno, clase positiva] o 0 [maligno, clase negativa] a partir de 30 características extraídas de personas reales. Es un problema de clasificación binaria con dos clases: 0 [clase negativa] y 1 [clase positiva].

```
1  from sklearn.datasets import load_breast_cancer
2  from sklearn.model_selection import train_test_split
3  from sklearn.linear_model import LogisticRegression
4  from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
5  from sklearn.metrics import recall_score, f1_score, roc_auc_score, roc_curve
6  import matplotlib.pyplot as plt
7
8  # Cargar el conjunto de datos
9  data = load_breast_cancer()
10 X, y = data.data, data.target
11
12 # Dividir el conjunto de datos en entrenamiento y prueba
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
14
15 # Crear el modelo de regresión logística
16 model = LogisticRegression(max_iter=10)
17
18 # Entrenar el modelo
19 model.fit(X_train, y_train)
20
21 # Realizar predicciones
22 y_pred = model.predict(X_test)
23
24 # Evaluar el modelo
25 print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

```
26  print(f'Precision: {precision_score(y_test, y_pred)}')
27  print(f'Recall: {recall_score(y_test, y_pred)}')
28  print(f'F1 Score: {f1_score(y_test, y_pred)}')
```

Esta vez sí hemos dividido el *dataset* original en un *trainset* [*X_train*, *y_train*] y un *testset* [*X_test*, *y_test*], en donde el segundo contiene[232] un 20% de las muestras originales. Por tanto vamos a entrenar un modelo con el 80% de las muestras y evaluarlo con el 20% restante.

El entrenamiento [*model.fit*] se lleva a cabo en la línea 19. El resultado es un modelo [*model*] que puede hacer predicciones [*predict*]. Y si podemos hacer predicciones, podemos evaluar el modelo usando el *testset*.

Pero antes recordemos un detalle, en este *dataset* la etiqueta 1 significa "maligno" y la etiqueta 0 "benigno". Es importante.

Empecemos por definir algunos conceptos, típicos de un clasificador binario:

- *True positive* [TP]: Las veces en las que el modelo afirma que es maligno [1] y en la realidad lo es.
- *True negative* [TN]: Las veces en las que el modelo afirma que es benigno [0] y realmente lo es.
- *False positive* [FP]: En donde el modelo predice maligno [1], pero realmente era benigno [0].
- *False negative* [FN]: En donde el modelo predice benigno [0], pero realmente se trataba de un cáncer maligno [1].

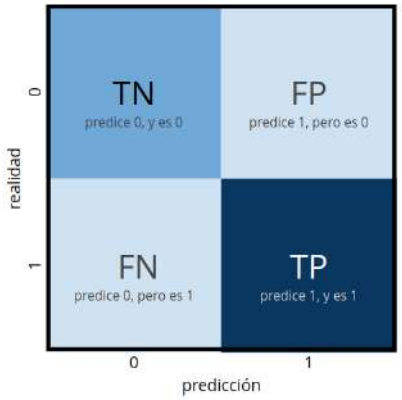



*Figura 70: Esquema general de matriz de confusión para un clasificador binario*

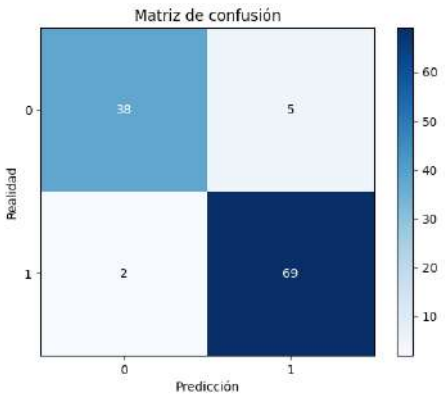



*Figura 71: Matriz de confusión, dataset: cancer_breast, algoritmo: regresión logística*

Para un clasificador binario, estos cuatro conceptos puede representarse en la **matriz de confusión.** En la figura 70 podemos ver el esquema general de una matriz de confusión con los términos que

232 *La división (split) se basa en un muestreo aleatorio. Tal y como está el código en este página, varias ejecuciones del mismo arrojará divisiones diferentes.*

acabamos de definir para un clasificador binario con etiquetas 0 y 1. En la figura 71 vemos la matriz de confusión generado usando la regresión logística sobre el *testset*.

Podemos observar que nuestro modelo se ha "confundido" 2 veces (FN) prediciendo benigno (0) cuando realmente fue maligno (1); y se ha equivocado 5 veces (FP) prediciendo maligno (1) cuando realmente era benigno (0).

¿Qué prefieres, ser diagnosticado como maligno y que finalmente no lo sea o que se te diagnostique un cáncer benigno cuando realmente es maligno? Los errores (las confusiones) en un clasificador binario pueden ser malas o menos malas, dependiendo del problema.

A partir de estos conceptos podemos sintetizar métricas que nos permitan evaluar el modelo y compararlo:

1. **Exactitud** (*accuracy*[233]): Se calcula dividiendo el número de predicciones correctas (TP+TN) entre el total de predicciones (TP+TN+FP+FN). Esta métrica nos da una idea general de que tan bien predice nuestro modelo, pero no hace diferencia entre predicciones malas o menos malas.

Desde el punto de vista positivo (TP):

2. **Precisión** (*precision*): Se centra en la calidad de las predicciones que el modelo hace para los tumores malignos (1). Se calcula como TP/(TP+FP). Es la proporción de la predicción correcta de "maligno" (1) respecto al total de predicciones "maligno" (1).

3. **Sensibilidad** (*recall*): Se centra en la capacidad del modelo para identificar todos los tumores malignos reales (1) en el conjunto de datos. Se calcula como TP/(TP+FN). Es la proporción de la predicción correcta de "maligno" (1) respecto al total de diagnósticos reales como "maligno" (1).

4. **F1**: Es una métrica que combina tanto la precisión como el *recall* en un solo número. Se utiliza especialmente en situaciones donde ninguna de estas las dos métricas es suficiente por sí sola para evaluar el rendimiento del modelo de forma efectiva. El F1 se define como la media armónica[234] de la precisión y el *recall*.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

   El F1 varía entre 0 y 1, siendo 1 el mejor valor posible y 0 el peor. Un F1 más alto indica un mejor equilibrio entre la precisión y el *recall*.

Y desde el punto de vista negativo (TN):

233 *A veces se traduce como "precisión", llevándonos a la confusión.*

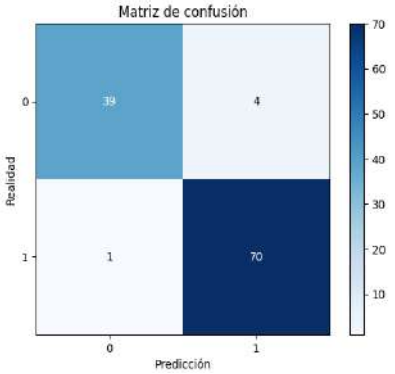
234 *Se utiliza la media armónica en lugar de la media aritmética porque da un valor más bajo cuando cualquiera de los términos (precisión o sensibilidad) es bajo. Esto es útil en situaciones donde se quiere que ambas métricas sean altas; un F1 alto solo se logra si ambas métricas son buenas.*

5. **Especificidad**: Equivalente a la precisión, pero con la clase negativa. Se centra en la calidad de las predicciones que el modelo hace para los tumores benignos (0). Se calcula como TN/(TN+FP).

6. **Valor predictivo negativo** (NPV, por sus siglas en inglés): Equivalente a *recall*, pero con la clase negativa. Se centra en la capacidad del modelo para identificar todos los tumores benignos reales (0) en el conjunto de datos. Se calcula como TN/(TN+FN).

7. **F1 negativo**: Es la métrica, equivalente al F1, que combina tanto la especificidad como el valor predictivo negativo en un solo valor.

Ahora podemos **comparar modelos**.




A la izquierda podemos ver la matriz de confusión que obtuvimos a partir de un modelo creado con el algoritmo SVM (ver el apartado 14.3.1). A primera vista podemos ver que FP (*false positive*) es 4 respecto a valor de 5 que obtuvimos con el modelo de regresión. Igualmente los *false negative* (FN) fueron 1 frente a 2. Esto nos indica que los valores de exactitud, precisión y *recall* van a ser mejores en el modelo SVM que en el modelo basado en regresión logística.

Calculando las métricas para los dos modelos, obtenemos la siguiente tabla:

| | SVM | Regresión logística |
|---|---|---|
| exactitud | 0.9561 | 0.9386 |
| precisión | 0.9459 | 0.9394 |
| sensibilidad | 0.9859 | 0.9718 |
| F1 | 0.9655 | 0.9517 |
| especificidad | 0.9070 | 0.8837 |
| NPV | 0.9750 | 0.9500 |
| F1 negativo | 0.9398 | 0.9157 |

Esta nos indica que el modelo SVM es ligeramente superior al de regresión logística usando los hiperparámetros indicados[235] al crear el modelo. Como en el contexto de nuestro problema (diagnóstico de biopsias de cáncer de mama) hay diagnósticos malos y menos malos, correspondientes a las clases 1 y 0 respectivamente, la métrica que nos permite comparar si cuando el modelo se confunde lo hace a favor de la paciente, en este caso es la especificidad y el NPV.

235 *Para la regresión logística indicamos el parámetro max_iter=10 (muy pocos). Si hubiésemos indicado 10000, este modelo igualaría al creado con el algoritmo SVM. Pero este último también tiene sus hiperparámetros, los cuales, al escoger los adecuados, mejorarían sus predicciones.*

## 16.3 Árboles de decisión

Vamos a continuar con el mismo *dataset*, pero con diferentes algoritmos. Seguimos dentro del aprendizaje supervisado aplicado a tareas de clasificación.

Los **árboles de decisión** son un algoritmo de aprendizaje automático utilizada tanto en tareas de clasificación como de regresión. Estos algoritmos se basan en la idea de construir un modelo en forma de un árbol, donde cada nodo interno representa una pregunta o prueba sobre una característica de los datos, y cada rama representa una posible respuesta a esa pregunta. Al final del árbol, encontramos las hojas que contienen las predicciones o las clases finales [ver el apartado 14.3.1].

El proceso de construcción de un árbol de decisión implica dividir el conjunto de datos de entrenamiento en subconjuntos más pequeños y más homogéneos con respecto a la variable objetivo. Esto se hace de manera recursiva hasta que se cumple algún criterio de parada, como, por ejemplo, la profundidad máxima del árbol.

Una vez que se ha construido el árbol de decisión, se puede utilizar para hacer predicciones sobre nuevos ejemplos. Para ello, se sigue el camino a través del árbol respondiendo las preguntas en cada nodo interno hasta llegar a una hoja, donde se encuentra la predicción correspondiente

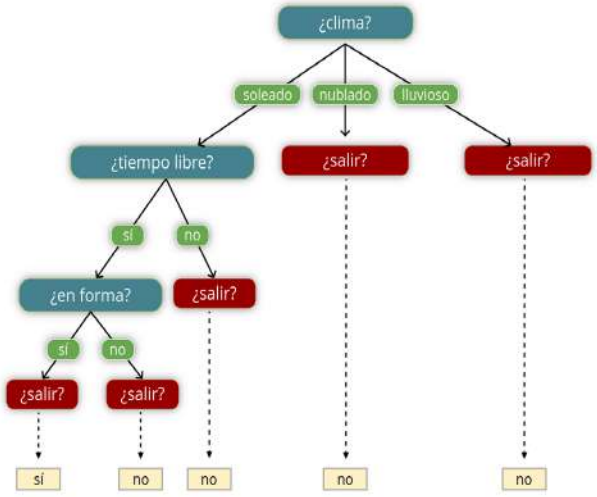



***Figura 72: Esquema de un árbol de decisión***

*Fuente: ChatGPT (concepto)*

Imaginemos que queremos predecir si una fruta es una manzana o una naranja, en función de su color. Para ello, el árbol de decisión podría tener un nodo raíz que examina el color de la fruta, y de este saldrían dos ramas: una para la fruta "roja" y otra para la fruta "naranja". Si seguimos por la rama de la fruta "roja", podríamos encontrar una hoja que clasifica la fruta como una "manzana".

De esta manera, el árbol ayuda a tomar decisiones basadas en las características de los datos.

La Figura 72 muestra un árbol de decisión que nos puede ayudar a tomar la decisión de "salir a correr o no"[236], con tres variables independientes [qué clima hace, tengo tiempo libre y estoy en forma] en forma de preguntas. La variable dependiente [la que queremos predecir] es "salir", igualmente en forma de pregunta. Los valores que puede tener esta columna son "si" y "no". Este árbol nos permite hacernos tres preguntas y siguiendo sus ramas llegar a una respuesta final, salgo o no salgo.

## 16.4 Bosques aleatorios

Los **bosques aleatorios** [*random forest*] son un algoritmo de aprendizaje automático basado en ensambles[237] de árboles de decisión. Combina múltiples árboles de decisión entrenados en diferentes subconjuntos de datos y características aleatorias para mejorar la precisión y reducir el sobreajuste.

*Random forest* utiliza un enfoque de muestreo aleatorio conocido como **muestreo con reemplazo** [*bootstrapping*], para crear diferentes conjuntos de datos de entrenamiento. Estos se utilizarán para entrenar árboles de decisión individuales.

> *El muestreo con reemplazo es una técnica estadística en la que se seleccionan elementos de una población o conjunto de datos de manera aleatoria y se devuelven al conjunto original antes de realizar una nueva selección. Esto significa que el mismo elemento puede ser elegido más de una vez durante el proceso.*

Como acabamos de decir, para cada conjunto de datos de entrenamiento generado se construye un árbol de decisión independiente. Una vez que todos los árboles de decisión han sido construidos, se utiliza el proceso de votación [clasificación[238]] o promedio [regresión[239]] para obtener la predicción final. Esta técnica se denomina *bagging* [acrónimo de *bootstrap aggregating*].

> ***Mi IA favorita dice**: Bagging es una técnica de ensamblado en aprendizaje automático que se utiliza para mejorar la estabilidad y la precisión de algoritmos de aprendizaje estadístico. Bagging se utiliza principalmente para reducir la variabilidad y evitar el sobreajuste.*

236 *Como indica la fuente, el concepto fue de ChatGPT, al igual que sus respuestas, que son poco saludables. Si te gusta salir a correr, sal siempre que quieras, tengas tiempo libre y un estado de forma aceptable como mínimo.*

237 *El diccionario de la Real Academia Española (RAE) incluye el término ensamble como sinónimo de ensambladura. Ambos conceptos se refieren al proceso y la consecuencia de ensamblar, un verbo que alude a ajustar, coordinar o acoplar algo.*

238 *En el caso de la clasificación, cada árbol emite una votación por la clase predicha, y la clase que recibe la mayoría de votos se considera la predicción final.*

239 *Para la regresión, se promedian las predicciones de todos los árboles para obtener el valor final.*

## 16.5 XGBoost

En el algoritmo anterior aprendimos lo que es hacer *bagging* con un algoritmo más simple (por ejemplo árboles de decisión), usando un conjunto de estos (ensamble) para sintetizar un resultado partiendo de sus predicciones. Ese es el funcionamiento de *random forest*. Usando un símil, como haría un jurado, promediando o usando la moda[240] a partir de sus opiniones individuales.

Pero esta no es la única estrategia que tenemos para potenciar algoritmos más simples y débiles.

El *boosting* es una técnica de ensamblado que tiene como objetivo mejorar la precisión de un modelo de predicción con menos capacidad. A diferencia del *bagging*, que entrena múltiples modelos de forma independiente y luego los combina, el *boosting* entrena múltiples modelos de forma secuencial, donde cada modelo intenta corregir los errores de los modelos anteriores. En cada iteración, se da más peso a las observaciones que fueron mal clasificadas o mal ajustadas por los modelos anteriores[241], forzando así al próximo modelo a centrarse en esos casos difíciles. Al final, las predicciones de todos los modelos se combinan para crear un modelo más robusto y preciso.

El *boosting* es especialmente efectivo cuando se utiliza con modelos débiles (como los árboles de decisión), es decir, modelos que realizan predicciones ligeramente mejores que una elección aleatoria. Algunos algoritmos populares de *boosting* incluyen AdaBoost, Gradient Boosting y **XGBoost**.

Vamos a resumir los resultados de aplicar todos estos algoritmos al *dataset*:

| | SVM (1) | Regresión logística | Árboles de decisión | Bosques aleatorios | XGBost (2) |
|---|---|---|---|---|---|
| **exactitud** | 0.9561 | 0.9386 | 0.9474 | 0.9649 | 0.9649 |
| **precisión** | 0.9459 | 0.9394 | 0.9577 | 0.9589 | 0.9589 |
| **sensibilidad** | 0.9859 | 0.9718 | 0.9577 | 0.9859 | 0.9859 |
| **F1** | 0.9655 | 0.9517 | 0.9577 | 0.9722 | 0.9722 |
| **especificidad** | 0.9070 | 0.8837 | 0.9302 | 0.9302 | 0.9302 |
| **NPV** | 0.9750 | 0.9500 | 0.9302 | 0.9756 | 0.9756 |
| **F1 negativo** | 0.9398 | 0.9157 | 0.9302 | 0.9524 | 0.9524 |

(1) Recordemos, visto en otro capítulo.

(2) Hemos hecho una búsqueda de los mejores hiperparámetros para este modelo.

Como podemos ver las métricas de los modelos se parecen mucho, de hecho la exactitud se sitúa entre el 93% y el 96% que, aún teniendo el cuenta el problema. No está mal.

El hecho de que modelos tan poderosos como los bosques aleatorios y XGBoost hayan dado los

240 *En estadística, la moda es el valor o valores que aparecen con más frecuencia en un conjunto de datos. A diferencia de la media y la mediana, la moda puede ser usada para describir tanto datos numéricos como categóricos.*

241 *Falsos negativos y falsos positivos en caso de clasificación binaria.*

mismos resultados, además teniendo en cuenta que en este último se usaron los hiperparámetros óptimos, nos puede hacer pensar que hemos llegado al límite de lo que podemos hacer con este *dataset*. O dicho de otras maneras, es el propio *dataset* el que impone un límite por ser demasiado simple[242].

## 16.6 Clasificación multiclase

La clasificación multiclase es una extensión de la clasificación binaria, donde el objetivo (la variable dependiente) es cualitativa y puede tomar más de dos valores o clases. Dado que ya se ha cubierto la clasificación binaria, en este apartado se presentarán varios métodos comúnmente usados para abordar problemas de clasificación donde hay más de dos clases. Los métodos varían en complejidad, desde técnicas que son extensiones naturales de los métodos binarios hasta algoritmos diseñados específicamente para el contexto multiclase.

El primer enfoque es usar clasificadores que, por su naturaleza, acepten clasificaciones multiclase de forma nativa. Por ejemplo los árboles de decisión y los algoritmos que los usan (*random forest* y *xgboost*, entre otros).

El segundo enfoque, si queremos usar un clasificador que no acepte más de dos clases, es convertir un problema de clasificación multiclase en varios problemas de clasificación binaria, y esto podemos hacerlo de dos formas:

- **Uno contra todos o contra el resto** (OvA/OvR, *One-vs-All/One-vs-Rest*). Consiste en entrenar un clasificador binario para cada clase, donde la clase en cuestión se etiqueta como positiva y todas las restantes se etiquetan como negativas. Por ejemplo: Imagina que tienes un *dataset* con imágenes de perros, gatos y pájaros, y tu problema es crear un clasificador para clasificar en estas tres clases. Con la estrategia OvA (o OvR) creas tres clasificadores:

  1. Un clasificador binario que detecta perros o no perros (gatos y pájaros).
  2. Un clasificador binario que detecta gatos o no gatos (perros y pájaros).
  3. Un clasificador binario que detecta pájaros o no pájaros (gatos y perros).

- **Uno contra Uno** (OvO, *One-vs-One*). Consiste en entrenar un clasificador binario para cada par de clases, donde cada clase se compara con cada una de las otras clases. Por ejemplo: De nuevo con el *dataset* con imágenes de perros, gatos y pájaros. Con la estrategia OvO creas tres clasificadores:

  - Un clasificador binario que detecta perros o gatos.
  - Un clasificador binario que detecta perros o pájaros.
  - Un clasificador binario que gatos o pájaros perros.

242 *Hay más interpretaciones, pero en este caso esta es la correcta.*

Ambos tiene pros y contras, en OvA tienes que entrenar tantos clasificadores como clases tengas [3 en el ejemplo] pero los datos estarán desequilibrados[243]; eso no ocurre con OvO pero debes entrenar $\frac{N(N-1)}{2}$ clasificadores[244].

La buena noticia es que el código te oculta todos estos detalles.

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC

model = SVC()

# Definir el clasificador
clf = OneVsRestClassifier(model)

# Entrenar el clasificador
clf.fit(X_train, y_train)

# Realizar predicciones
y_pred = clf.predict(X_test)
```

Fíjate. Sólo tienes que instanciar el algoritmo [SVC en este caso] y crear un objeto [*OneVsRestClassifier*, en negritas] que, usando el modelo que le pasas, creará tantos clasificadores como necesite y los entrenará con los datos apropiados. Desde tu punto de vista sólo tienes que saber qué haces, pero no cómo. Eso lo hace la biblioteca de Python.

Ahora necesitamos un *dataset* que posea múltiples clases. Vamos a usar uno clásico que, aunque esté orientado a imágenes y más susceptible de ser usado en *deep learning*, bien puede usarse en este contexto. Me refiero al MNIST de dígitos.

El conjunto de datos MNIST [*Modified National Institute of Standards and Technology*] Digits es uno de los *datasets* más populares en el mundo del aprendizaje automático. Es una colección de 70.000 imágenes, de 28x28 en escala de grises, de dígitos escritos a mano y se utiliza comúnmente para entrenar modelos de reconocimiento de imágenes y como un punto de referencia para nuevos algoritmos. En nuestro caso lo usaremos para comprobar cómo entrenar modelos que estamos viendo y evaluarlos.

Pero, las imágenes son de dos dimensiones y estamos trabajando con datos tabulados en columnas [cada columna correspondiente a cada variable independiente sería una dimensión]. ¿Cómo convertimos uno en otro? Es muy fácil, y de paso nos demuestra la potencia de estos algoritmos que son agnósticos respecto a información o conocimiento, ¡sólo ven números!

243 *El desequilibrio o desbalanceo de las clases de los datos ocasiona sesgos, al estar una clase menos representada que el resto.*

244 *En nuestro caso igualmente 3 ( (3*2)/2 ), pero imagina si tienes 10 clases, entonces deberías entrenar 45 clasificadores, mientras que con OvA sólo 10.*

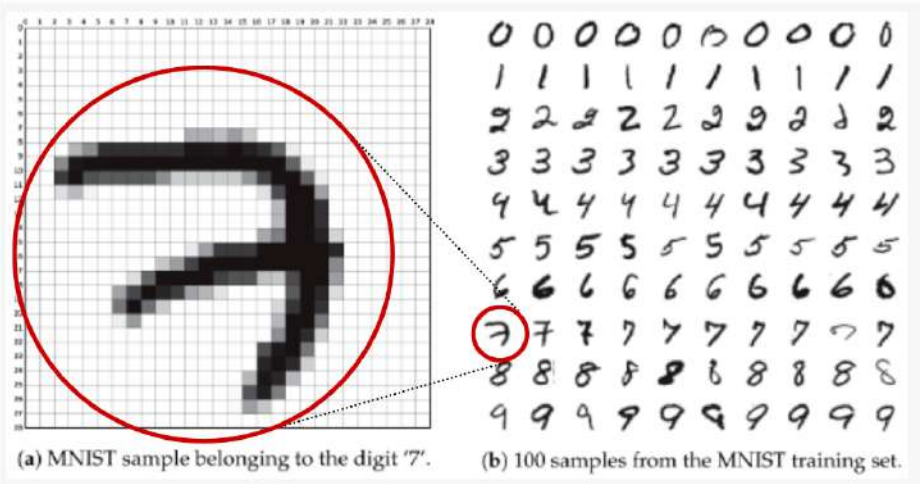



*Figura 73: Muestra del dataset MNIST Digits*

Cada una de las 70K imágenes del *dataset* está formada por 28x28 [784] pixels, cada uno de estos codificando un valor de intensidad o tono de gris [0 negro, 255 blanco]. Así que lo que hay que hacer es "aplanar" la imagen.


*Figura 74: Ejemplo de aplanamiento de una imagen en 2D (5x5) a una forma en 1D (1x25)*

Esta acción convierte una imagen de 28x28 en un vector de 1x768, esto es 768 números. Cada uno de estos números es una de las variables independientes. Ya tenemos la forma de convertir cada imagen a un formato tabular, como la de la Figura 61.

El código que sigue carga los datos [de la librería *tensorflow*], que ya se nos entregan divididos en datos de entrenamiento [*train_img* y *train_lbl*, 60k muestras] y datos de testeo [*test_img* y *test_lbl*, 10k muestras]. Seguidamente se lleva a cabo el dimensionamiento de cada una de las 70k imágenes.

```
1   import tensorflow as tf
2
3   # Cargar el conjunto de datos Fashion-MNIST
4   (train_img, train_lbl), (test_img, test_lbl) = tf.keras.datasets.mnist.load_data()
```

```
5
6   # Redimensionar las imágenes: de 28x28 a 784 (28x28) elementos en un único array
7   train_img = train_img.reshape(-1, 28 * 28)
8   test_img = test_img.reshape(-1, 28 * 28)
9
10  # Normalizar los valores de los píxeles en el rango [0, 1]
11  train_img = train_img.astype('float32') / 255
12  test_img = test_img.astype('float32') / 255
```

Finalmente se dividen cada uno de los pixels de todas las imágenes entre 255, esto ocasiona que los números que va a manejar el modelo se encuentran entre 0.0 y 1.0; la razón de esta normalización la encontramos en la mejora de la estabilidad numérica[245].

**Definiendo el problema**: Queremos hacer una clasificación multiclase, de tal forma que, a partir de una imagen 28x28 de un número del 0 al 9 escrito a mano, obtengamos el dígito que está escrito. Esto es, un reconocimiento óptico de dígitos.

En el *notebook* del capítulo puedes encontrar todo el código que hace esto, aquí solo mostraré los resultados y como se evalúan los modelos multiclase.

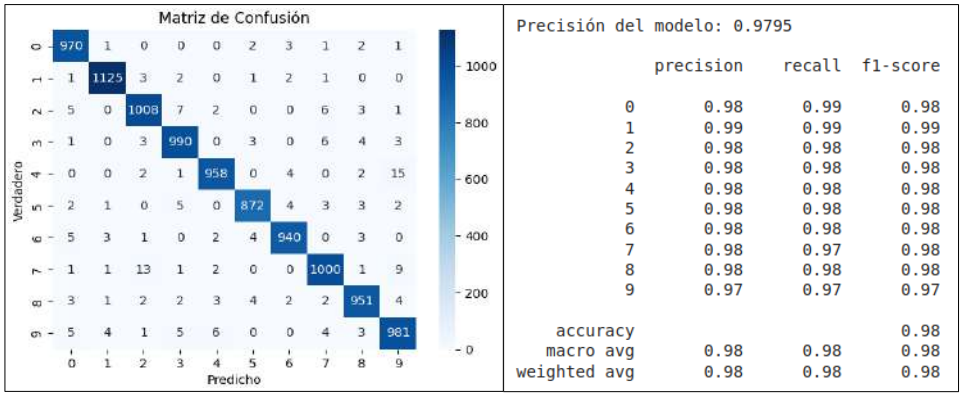



*Figura 75: Matriz de confusión multiclase 10x10 y métricas*

En la evaluación de modelos multiclase también tenemos matrices de confusión, pero en vez de ser 2x2 (en un clasificador binario) son *nxn* (para un clasificados de *n* clases).

En la Figura 75 podemos observar el resultado de la evaluación del modelo creado con el algoritmo de regresión logística que vimos anteriormente. Como podemos ver ahora tenemos 10 clases (0-9) en dos ejes: "Predicho" (abscisas) y "Verdadero" (coordenadas). También debemos observar la diagonal, en donde se encuentran los ciertos: para una predicción del dígito 0 se acertaron 970 veces y se fallaron 23 veces.

Observa, los errores más comunes son predecir un 2 y que realmente sea un 7 (13 veces) y predecir un 9 y que realmente sea un 4 (15 veces). Vuelve a la Figura 73 para ver los parecidos entre

245 *Las operaciones que involucran números flotantes pueden sufrir de inestabilidad numérica. Mantener los números en un rango razonable (como 0.0-1.0) puede ayudar a mitigar este problema. En otros modelos este tipo de normalizaciones tienen otros propósitos, como hacer que todas las características compartan el mismo rango.*

estos dígitos.

¿Y respecto a las métricas? Las más interesantes son *precision* y *recall* por dígito, además de su F1.

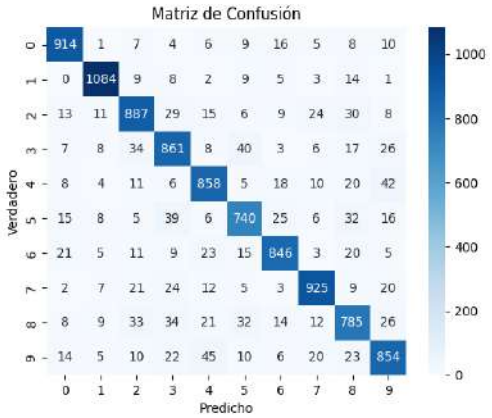
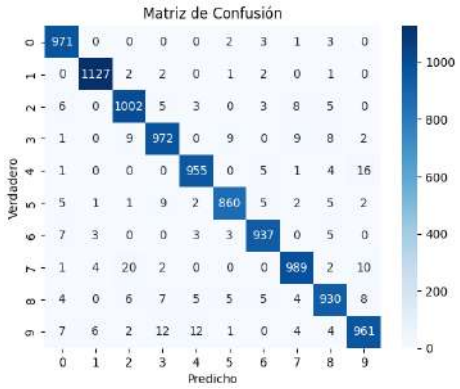



*Figura 76: Matriz de confusión multiclase 10x10 con un algoritmo árbol de decisión*

En la Figura 76 tenemos la misma estructura de información pero con los datos de un modelo basado en árbol de decisión. Como ya indicamos este algoritmo acepta de forma nativa clasificación multiclase. No usa OvA o OvO.

La precisión baja hasta un 87.54 de media, muy por debajo del óptimo. En efecto, este *dataset* es más difícil para un árbol de decisión. Todos los dígitos tienen bastantes errores de predicción, sólo se salva el dígito 1, con una precisión del 95% y un *recall* de 96% respectivamente.

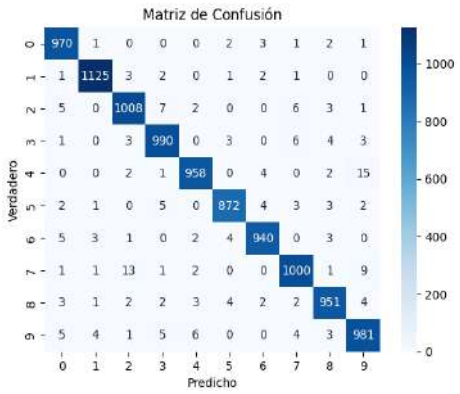
Como indicamos anteriormente, a partir del algoritmo de árbol de decisión podemos aplicar una estrategia de ensamble, bien *bagging* [*random forest*] o *boosting* [*xgboost*], para potenciar su funcionamiento. En la Figura 77 y la Figura 78 podemos ver respectivamente sus métricas y matrices de confusión. Hemos mejorado, pero ¿se podrá hacer mejor?




Precisión del modelo: 0.9704

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 |
| 1 | 0.99 | 0.99 | 0.99 |
| 2 | 0.96 | 0.97 | 0.97 |
| 3 | 0.96 | 0.96 | 0.96 |
| 4 | 0.97 | 0.97 | 0.97 |
| 5 | 0.98 | 0.96 | 0.97 |
| 6 | 0.98 | 0.98 | 0.98 |
| 7 | 0.97 | 0.96 | 0.97 |
| 8 | 0.96 | 0.95 | 0.96 |
| 9 | 0.96 | 0.95 | 0.96 |
| accuracy | | | 0.97 |
| macro avg | 0.97 | 0.97 | 0.97 |
| weighted avg | 0.97 | 0.97 | 0.97 |

*Figura 77: Matriz de confusión multiclase 10x10 y métricas con un algoritmo de random forest*

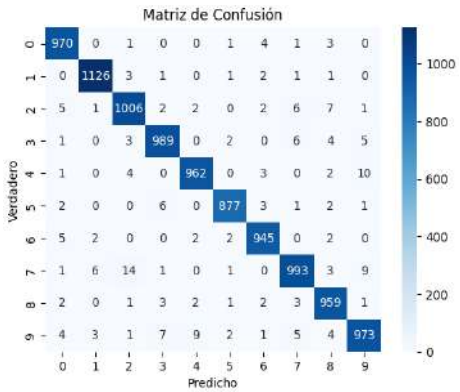



Precisión del modelo: 0.9795

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.98 | 0.99 | 0.98 |
| 1 | 0.99 | 0.99 | 0.99 |
| 2 | 0.98 | 0.98 | 0.98 |
| 3 | 0.98 | 0.98 | 0.98 |
| 4 | 0.98 | 0.98 | 0.98 |
| 5 | 0.98 | 0.98 | 0.98 |
| 6 | 0.98 | 0.98 | 0.98 |
| 7 | 0.98 | 0.97 | 0.98 |
| 8 | 0.98 | 0.98 | 0.98 |
| 9 | 0.97 | 0.97 | 0.97 |
| accuracy | | | 0.98 |
| macro avg | 0.98 | 0.98 | 0.98 |
| weighted avg | 0.98 | 0.98 | 0.98 |

*Figura 78: Matriz de confusión multiclase 10x10 y métricas con un algoritmo XBoost*

Ahora vamos a aprobar con SVM [*Support vector machines*]. Pero este algoritmo, como ya hemos indicado, no soporta multiclases [de forma nativa es un clasificador binario[246]], así que tenemos que usarlo mediante una estrategia que convierta un clasificador binario en otro multiclase, como OvA o OvO. De hecho probaremos ambos, para poder comparar.




Precisión del modelo: 0.9800

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.98 | 0.99 | 0.98 |
| 1 | 0.99 | 0.99 | 0.99 |
| 2 | 0.97 | 0.97 | 0.97 |
| 3 | 0.98 | 0.98 | 0.98 |
| 4 | 0.98 | 0.98 | 0.98 |
| 5 | 0.99 | 0.98 | 0.99 |
| 6 | 0.98 | 0.99 | 0.98 |
| 7 | 0.98 | 0.97 | 0.97 |
| 8 | 0.97 | 0.98 | 0.98 |
| 9 | 0.97 | 0.96 | 0.97 |
| accuracy | | | 0.98 |
| macro avg | 0.98 | 0.98 | 0.98 |
| weighted avg | 0.98 | 0.98 | 0.98 |

*Figura 79: Matriz de confusión multiclase 10x10 y métricas para OvA y OvO con SVM*

El resultado ha sido el mismo para ambas estrategias, sólo se diferenciaron en el tiempo que emplearon para entrenar el modelo. ¿Por qué? Bien, hay varias razones que lo explican, pero las dos más plausibles son [1] los datos son muy separables, como es el caso, y cuando se equivocan lo hacen sobre la mismas predicciones; [2] SVM es muy potente, y lo que hace lo hace muy bien, así que es posible que den el mismo resultado.

Parece que hemos mejorado respecto a los modelos creados anteriormente, ya que tenemos una precisión media de 98%, siendo la peores predicciones las que se corresponden a los dígitos 2, 7 y 9. Las máximas confusiones que el modelo predice son los pares [2, 7] y [9, 4], como era de esperar.

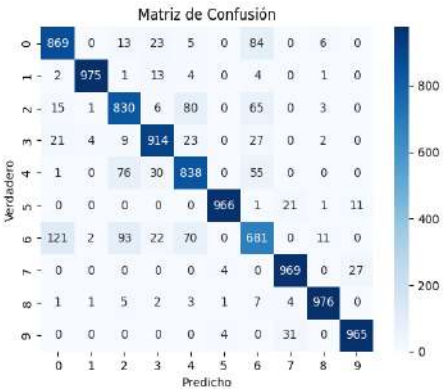
246 *Recordemos, SVM calcula los "pasillos" que separan dos clases, no puede hacer "pasillos" que separen tres o más (ver apartado 14.3.1). Estos "pasillos" los llamamos hiperplanos.*

> ***Conclusiones**: Los algoritmos de aprendizaje automático son muy buenos creando modelos que predigan un valor (regresión) o una clase (clasificación), siempre y cuando el dataset cumpla, principalmente, con: criterios de calidad y cantidad (ausencia de sesgos, diversidad, completud, etc.) e independencia.*

Dependemos de los datos. Para mostrar este hecho, vamos a repetir todos los cálculos que hemos hecho en "clasificación multiclase", pero con otro dataset de idénticas proporciones pero diferente contenido.


*Figura 80: Conjunto de datos MNIST Fashion (parcial)*

El *dataset* MNIST *fashion* tiene el mismo tamaño que el *dataset* MNIST *digits:* 70k muestras, cada una de ellas imágenes de 28x28 pixels en tonos de grises y 10 clases. Sin embargo el contenido de las imágenes no es el mismo y las clases no significan lo mismo.




Precisión del modelo: 0.8983

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.84 | 0.87 | 0.86 |
| 1 | 0.99 | 0.97 | 0.98 |
| 2 | 0.81 | 0.83 | 0.82 |
| 3 | 0.90 | 0.91 | 0.91 |
| 4 | 0.82 | 0.84 | 0.83 |
| 5 | 0.99 | 0.97 | 0.98 |
| 6 | 0.74 | 0.68 | 0.71 |
| 7 | 0.95 | 0.97 | 0.96 |
| 8 | 0.98 | 0.98 | 0.98 |
| 9 | 0.96 | 0.96 | 0.96 |
| accuracy | | | 0.90 |
| macro avg | 0.90 | 0.90 | 0.90 |
| weighted avg | 0.90 | 0.90 | 0.90 |

*Figura 81: Matriz de confusión multiclase 10x10 y métricas para XGBoost sobre MNIST fashion*

Este *dataset* contiene imágenes en baja resolución de 10 tipos de ropa [camisetas, pantalones, jer-

séis, vestidos, abrigos, sandalias, camisas, zapatillas, bolsos y botines]. Es un reto. Observa la Figura 80, diferenciar bolsos del resto es fácil, igual que sandalias y botines del resto, pero ¿diferencias bien sandalias y botines en las imágenes, camisetas y abrigos, etc.? En este caso los datos de los que partimos son de menor calidad por su baja resolución.

La Figura 81 muestra las métricas del mejor modelo: una precisión de menos del 89.8%.

Para finalizar este apartado sobre el aprendizaje supervisado, vamos a retomar un efecto pernicioso que debemos evitar a todas costa. **El sobreentrenamiento** [*overfitting*]. Este concepto ya lo tratamos brevemente al principio del capítulo, aquí vamos a profundizar en él.

El sobreentrenamiento es una situación común en el aprendizaje automático donde un modelo aprende demasiado bien a partir de los datos de entrenamiento pero generaliza mal ante nuevas observaciones. Podemos decir que aprende "memorizando" en lugar de generalizando.

Vamos a presentar dos modelos, entrenados sobre un *dataset* generado para la ocasión.

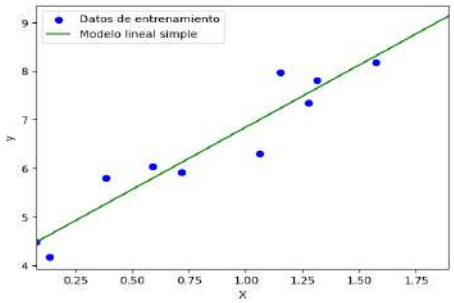



*Figura 82: Dataset aleatorio modelado mediante regresión simple*

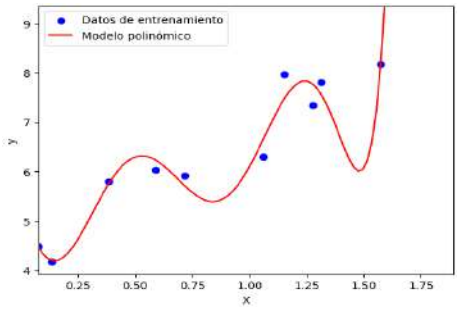



*Figura 83: Dataset aleatorio modelado mediante regresión polinómica*

El primero [Figura 82] arroja un MAE [respecto al *trainset*] de 0.3404 y el segundo [Figura 83] un MAE de 0.2071 ¿qué modelo se ajusta mejor a los datos con los que se ha entrenado? El modelo de regresión polinómico[247] se ajusta bastante bien, casi un 40% mejor que la versión simple. Que obtengas un modelo casi un 40% mejor que otro es una buena noticia .. o no.

Como ya hemos dicho, tenemos que evaluar los modelos respecto al futuro: respecto a datos que aún no tenemos, y como las máquinas del tiempo están caras, debemos emplear técnicas[248] que "simulen" tener datos del futuro. En nuestro caso entrenar y evaluar con conjuntos de datos disjuntos: *trainset* y *dataset*. Entrenamos con el primero y evaluamos con el segundo. Vamos a ver estas gráficas, pero con el conjunto de testeo [Figura 84 modelo de regresión simple con los datos de testeo y Figura 85 modelo de regresión polinómica con los datos de testeo].

247 *De grado 7, por cierto.*

248 *Hay más técnicas (por ejemplo evaluar usando k-fold durante el entrenamiento), pero esta es la más simple y sus conceptos son la base de casi todas las demás.*

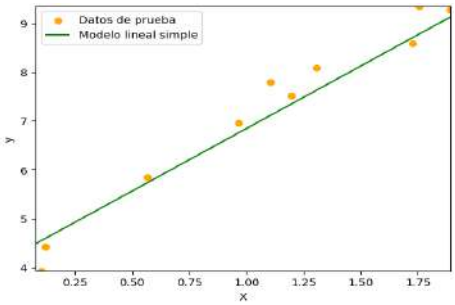



*Figura 84: Dataset aleatorio modelado mediante regresión simple*

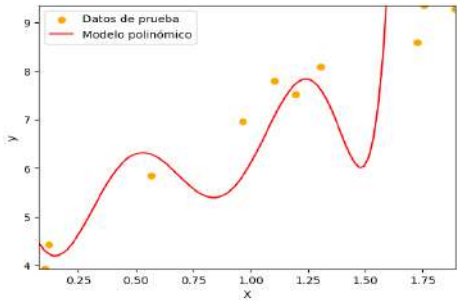



*Figura 85: Dataset aleatorio modelado mediante regresión polinómica*

¿Cual ajusta mejor ahora? El MAE para la regresión lineal simple simple (izquierda) arroja un valor de 0.3267, mientras que la regresión polinómica, un 40% mejor si la evaluamos sólo con los datos de entrenamiento, ahora arroja un MAE de 20.6026 usando los "datos del futuro", el *testset*. ¡Un 99% peor!

Pero ¿no habrá mucha diferencia en la predicción al usar sólo una parte del conjunto de entrenamiento y dejar otra parte – intocable mientras se entrena – para la evaluación? Veámoslo: la diferencia entre los MAE de la regresión lineal simple usando el *trainset* y el *testnet* es sólo de un 4%. Nuestro modelo, a priori, sólo se equivoca un 4% cuando se enfrenta a datos no vistos durante el entrenamiento.

¿Y qué tiene que ver esto con el *overfitting*?

> ***Atención**: una de las formas más efectivas de detectar el overfitting es calcular la diferencia de las métricas cuando se usa el conjunto de entrenamiento y el conjunto de testeo. Básicamente un modelo que está sobreentrenando se comporta muy bien en datos del conjunto de entrenamiento, pero muy mal en datos del conjunto de testeo*[249].

En nuestro ejemplo el modelo lineal simple tenía un porcentaje del 4% de pérdida de acierto (media de los errores absolutos) entre los conjuntos de *trainset* y *testset*. El modelo polinómico obtuvo un 99% de pérdida. ¿Por qué? Porque este último cayó en el sobre-entrenamiento u ***overfitting***.

Algunos de los algoritmos son propensos a crear modelos ya que, por su naturaleza, tienden a "memorizar" los datos del *dataset* durante el entrenamiento, ocasionando que a la hora de predecir con datos que no ha visto anteriormente obtengan unos errores muy superiores. Por ejemplo los árboles de decisión.

249 *Igual que un alumno que aprende de memoria la lección, pero no la entiende.*

## 16.7 Agrupamiento

Este apartado se centra en un caso particular dentro de la familia de algoritmos que no necesitan etiquetas para aprender a partir de los datos.

> ***Recordemos**: en la Figura 61 vimos que los dataset, entendidos estos como muestras[250] y características[251], son la fuente del aprendizaje cuyo fin es crear un modelo que, a partir de las variables independientes, pueda generalizar y predecir el valor de la etiqueta que le corresponde. Esto se llama aprendizaje supervisado porque tiene que existir al menos una columna con la solución[252] para poder aprender.*

El aprendizaje supervisado, que es el que hemos visto hasta ahora en este capítulo, se denomina "supervisado" precisamente porque necesita la variable dependiente u objetivo con la que se "supervisa" el entrenamiento.

En este apartado hablaremos del **aprendizaje no supervisado**, en el cual el *dataset* no tiene una columna o variable objetivo. Todas son variables independientes, o al menos son tratas así[253].

Y dentro del aprendizaje no supervisado vamos a ver la técnica de agrupamiento (o *clustering*). El agrupamiento es una técnica dentro del aprendizaje no supervisado que tiene como objetivo segmentar un conjunto de datos en grupos o clústeres homogéneos. En otras palabras, busca agrupar elementos de manera que estos, dentro de un mismo grupo, sean más similares entre sí que con los elementos de otros grupos.

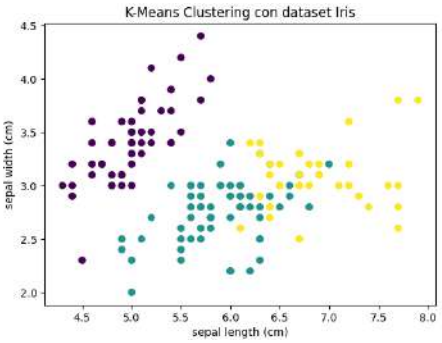



*Figura 86: Ejemplo de clustering con el dataset Iris en 2D.*

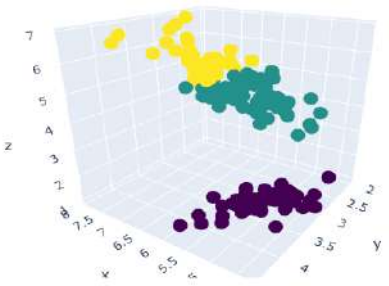



*Figura 87: Ejemplo de clustering con el dataset Iris en 3D.*

En la Figura 86 podemos ver una representación de un agrupamiento realizado con el *dataset* Iris

250 *Las filas.*

251 *Las columnas, también llamadas variables o características. Estas podemos dividirlas en dos tipos: variables independientes y variable dependiente (objetivo).*

252 *La variable dependiente, u objetivo. Puede haber más de una, pero aquí no lo estamos contemplando.*

253 *Un pequeño apunte: es muy normal usar un dataset típico de aprendizaje supervisado y usarlo para aprendizaje no supervisado, eso sí eliminando (pero no olvidando) la variable dependiente.*

en dos dimensiones. Al estar en 2D parece que dos de los clústeres solapan, pero en la Figura 87, en 3D interactiva [ver el *notebook* correspondiente], he intentado poner la perspectiva en la que podemos ver con más claridad que, aunque compartan frontera, apenas solapan.

El código que entrena el modelo es muy simple:

```
# Utilizar K-Means para encontrar 3 clusters
kmeans = KMeans(n_clusters=3)
kmeans.fit(data)

# Obtener las etiquetas de los clusters y los centroides
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

En donde la variable *data* contiene el *dataset* Iris sin la columna "*species*", usa un algoritmo llamado **KMeans** el cual obtiene – después de entrenar con *fit* – las etiquetas [*labels*] y los centroides [*centroids*]. Esto da lugar a dos preguntas: ¿qué son las etiquetas?

El resultado de un modelo de *clustering* es un conjunto de etiquetas, tantas como filas tiene el *dataset*, en donde la etiqueta *n* indica a qué clúster pertenece la fila *n*.

¿Qué son los centroides? Explicar que son los centroides nos da la oportunidad para explicar cómo funciona KMeans.

Antes de empezar necesitamos definir un concepto: **similitud**. Para llevar a cabo el agrupamiento, es necesario definir alguna métrica que mida la similitud entre los puntos de datos. En capítulo apartado 10.3, y en apartados anteriores a este, vimos métricas de similitud y distancia entre textos. Ahora básicamente necesitamos medir de alguna forma una distancia entre las muestras del *dataset*. Si este tiene *n* columnas y cada columna contiene números, podemos ver cada fila como un vector de *n* componentes. De esta manera podemos ver cada muestra como un punto de *n* dimensiones. ¿cómo calcular la distancia entre puntos en un espacio de *n* dimensiones?

$$Distancia\ Euclídea(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

La distancia Euclídea puede ser extendida para calcular un valor numérico [distancia] a partir de dos puntos en un espacio de *n* dimensiones; y es la manera que por defecto[254] utiliza KMeans.

Ya estamos en disposición de ver como funciona este algoritmo para crear modelos de *clustering*:

1. Seleccionados *c* puntos como centroides iniciales. Cada centroide representa a un clúster y por tanto tiene su propia etiqueta.
2. Para el resto de puntos: asigna cada uno de ellos al clúster cuyo centroide sea el más cercano, según la métrica que estemos usando.

254 *Otras métricas de similitud o distancia son: distancia Manhattan, similitud del coseno, distancia Jaccard y la distancia Mahalanobis; entre otras. Observar que se pueden mediar las distancias entre textos según su similitud (Capítulo 10). También es posible calcular la similitud entre imágenes.*

3. Actualiza los centroides: actualizamos los centroides re-calculándolos en función de los puntos del clúster al que representan.
4. Repite los pasos 2 y 3 hasta que los centroides apenas cambien.

El Vídeo 13 es una animación en donde se muestra este entrenamiento. La única duda que podría plantearse es ¿por qué se mueven los centroides? O, mejor ¿por qué al actualizar los centroides, estos se mueven? Para re-calcular el centroide j-ésimo se toman todos los puntos del clúster *j* y se calcula:

$$\mu(j)=\frac{1}{|C(j)|}\sum_{i\in C(j)} x_i$$

Parece complicado pero en realidad es muy fácil: para cada clúster, sumamos los puntos que pertenecen al dicho clúster y el resultado lo dividimos entre en número de puntos en el clúster. ¡Es una media! Por ejemplo, supongamos que el clúster *j* está formado por los siguientes 5 puntos de 3 dimensiones:

- P1=[1, 2, 3]
- P2=[4, 5, 6]
- P3=[7, 8, 9]
- P4=[10, 11, 12]
- P5=[13, 14, 15]

El punto suma resultante es [35, 40, 45] y al dividir este entre 5, obtenemos [7,8,9][255]. El nuevo centroide. De esta forma, cada vez que hacemos el punto [2] tenemos que recalcular los centroides y estos cambian en [3]. Finalizamos cuando los centroides no cambien porque en el punto [2] no hay cambios en los clústeres.

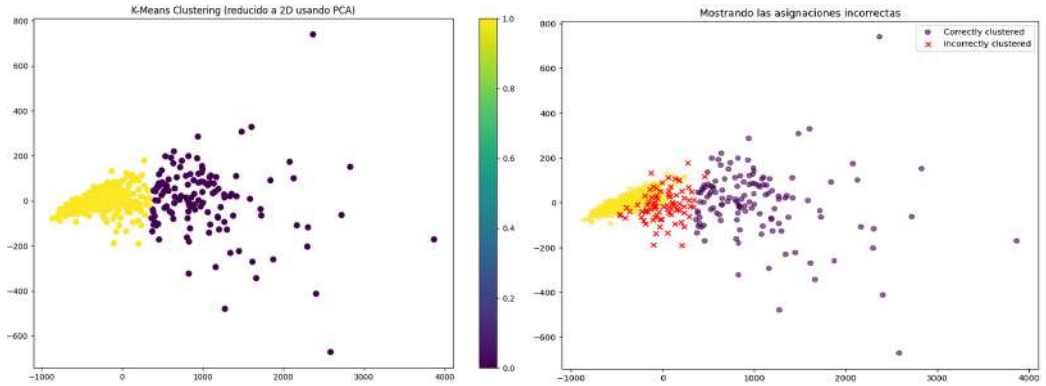
***Vídeo 13: Animación del algoritmo KMeans durante el entrenamiento***

El agrupamiento de datos se utiliza para entrenar modelos que tengan la capacidad de aprender sin supervisión a diferenciar los datos por medio de algún patrón – agrupándolos – para, posteriormente, ante la presencia de nuevos datos clasificarlos en algunos de estos grupos.

Volvamos al *dataset* de cáncer de pecho, saquemos la columna objetivo que indica si es benigno o no y hagamos una clusterización de los datos, a ver si podemos clasificar aun así los datos en ma-

255 *Que sea igual que uno de los puntos es mera coincidencia.*

lignos o benignos. Las siguientes imágenes muestran la clusterización de las 30 variables independientes, proyectadas en 2D por medio de un algoritmo de reducción de la dimensionalidad, a la izquierda agrupadas en dos clústeres. Es importante recalcar que al algoritmo de clusterización sólo le fue indicado cuantos clústeres queríamos tener, fue el propio algoritmo el que estableció el criterio para separar ambos grupos.




A la derecha tenemos la misma agrupación, pero ahora destacando en rojo las asignaciones incorrectas que ha realizado la clusterización basándose sólo en los datos y ayudado, ahora sí, por la columna objetivo que originalmente hemos retirado del *dataset* para usarla como forma de evaluar la clusterización.

> ***Nota**: la clusterización en sí ha sido capaz de separar en dos grupos las muestras sólo basándose en los patrones internos de las mismas, no en la columna objetivo, la cual sólo se usó para evaluar el modelo y mostrar los errores de asignación con una intención pedagógica.*

# RETOS DEL CAPITULO 16

1. Investiga y describe en tus propias palabras qué es el "entrenamiento de modelos" en el contexto del aprendizaje automático.
2. Interroga a tu IA favorita, cuan importante es el conjunto de datos en el aprendizaje automático.
3. ¿Cuál es la diferencia entre un conjunto de entrenamiento y un conjunto de prueba? ¿Por qué son necesarios ambos? Intenta deducirlo sin ayuda externa.
4. Investiga y explica qué es la validación cruzada y cómo se utiliza en la evaluación de modelos.
5. ¿Qué es la matriz de confusión y para qué se utiliza en la evaluación de modelos de clasificación?
6. Dialoga con tu agente conversacional favorito, ¿Qué significa el término "sobreajuste" en el contexto del aprendizaje automático? ¿Cómo se puede evitar?
7. Investiga y explica la diferencia entre regresión y clasificación en el aprendizaje automático.
8. ¿Qué son los datos abiertos? Busca sitios *web* públicos de tu ayuntamiento o comunidad en donde compartan datos.
9. Repite el reto anterior, pero a nivel nacional.
10. Investiga cómo se elige el algoritmo adecuado para entrenar un modelo a partir de unos datos dados.
11. Investiga y describe el concepto de "conjunto de validación" y su papel en el ajuste de hiperparámetros.
12. Dialoga con tu IA favorita, qué es y para que sirve una curva ROC. Indícale tu nivel de conocimientos para que no sea muy técnico.
13. Investiga ejemplos de al menos dos técnicas de reducción de dimensionalidad utilizadas en el aprendizaje automático. ¿Por qué las usamos?
14. Realiza un experimento de entrenamiento y evaluación de un modelo de regresión lineal utilizando una herramienta en línea basados en *notebooks*.