2_Evaluacion de modelos

July 6, 2025

Creado por:

Isabel Maniega

[1]:
```
from IPython import display
```

1 Reconocer la importancia de los conjuntos de datos de prueba en la evaluación de modelos.

3 métodos para dividir conjuntos de datos de aprendizaje automático

Existen varios métodos para dividir conjuntos de datos para modelos de aprendizaje automático. El enfoque correcto para la división de datos y la proporción de división óptima dependen de varios factores, incluidos el caso de uso, la cantidad de datos, la calidad de los datos y la cantidad de hiperparámetros.

Muestreo aleatorio (Random Sampling)

El método más común para dividir un conjunto de datos es el muestreo aleatorio. Como sugiere el nombre, el método implica mezclar el conjunto de datos y asignar aleatoriamente muestras a conjuntos de entrenamiento, validación o prueba según proporciones predeterminadas. Con conjuntos de datos con equilibrio de clases, el muestreo aleatorio garantiza que la división sea imparcial.

Si bien el muestreo aleatorio es el mejor enfoque para muchos problemas de aprendizaje automático, no es el enfoque correcto con conjuntos de datos desequilibrados . Cuando los datos consisten en proporciones de clases sesgadas, el muestreo aleatorio casi con certeza creará un sesgo en el modelo.

División de conjuntos de datos estratificados (Stratified Dataset Splitting)

La división estratificada de conjuntos de datos es un método que se utiliza habitualmente con conjuntos de datos desequilibrados, en los que ciertas clases o categorías tienen significativamente menos instancias que otras. En tales casos, es fundamental garantizar que los conjuntos de entrenamiento, validación y prueba representen adecuadamente la distribución de clases para evitar sesgos en el modelo final.

En la división estratificada, el conjunto de datos se divide conservando las proporciones relativas de cada clase en las divisiones. Como resultado, los conjuntos de entrenamiento, validación y prueba contienen un subconjunto representativo de cada clase, manteniendo la distribución de clases original. Al hacerlo, el modelo puede aprender a reconocer patrones y hacer predicciones para todas las clases, lo que da como resultado un algoritmo de aprendizaje automático más sólido y confiable.

División de validación cruzada (Cross-Validation Splitting)

El muestreo de validación cruzada es una técnica que se utiliza para dividir un conjunto de datos en conjuntos de entrenamiento y validación con fines de validación cruzada. Implica la creación de múltiples subconjuntos de los datos, cada uno de los cuales sirve como conjunto de entrenamiento o conjunto de validación durante diferentes iteraciones del proceso de validación cruzada.

La validación cruzada de K-fold y la validación cruzada de K-fold estratificada son técnicas comunes. Al utilizar estas técnicas de muestreo de validación cruzada, los investigadores y los profesionales del aprendizaje automático pueden obtener métricas de rendimiento más confiables e imparciales para sus modelos de aprendizaje automático, lo que les permite tomar decisiones mejor informadas durante el desarrollo y la selección de modelos.

3 errores que se deben evitar al dividir datos

Existen algunos errores comunes que los científicos de datos y los ingenieros de ML cometen al dividir conjuntos de datos para el entrenamiento de modelos.

Tamaño de muestra inadecuado

Un tamaño de muestra insuficiente en los conjuntos de entrenamiento, validación o prueba puede generar métricas de rendimiento del modelo poco confiables. Si el conjunto de entrenamiento es demasiado pequeño, el modelo puede no capturar suficientes patrones o no generalizar bien. De manera similar, si el conjunto de validación o el conjunto de prueba son demasiado pequeños, la evaluación del rendimiento puede carecer de significancia estadística.

Fuga de datos

La fuga de datos se produce cuando la información del conjunto de validación o del conjunto de prueba se filtra inadvertidamente en el conjunto de entrenamiento. Esto puede generar métricas de rendimiento demasiado optimistas y una sensación exagerada de la precisión del modelo final. Para evitar la fuga de datos, es fundamental garantizar una separación estricta entre el conjunto de entrenamiento, el conjunto de validación y el conjunto de prueba, asegurándose de que no se utilice información de los conjuntos de evaluación durante el entrenamiento del modelo.

Mezcla o clasificación inadecuada

Mezclar u ordenar incorrectamente los datos antes de dividirlos puede generar sesgos y afectar la generalización del modelo final. Por ejemplo, si el conjunto de datos no se mezcla aleatoriamente antes de dividirlo en un conjunto de entrenamiento y un conjunto de validación, puede generar sesgos o patrones que el modelo puede aprovechar durante el entrenamiento. Como resultado, el modelo entrenado puede sobreajustarse a esos patrones específicos y no generalizarse bien a datos nuevos que no se hayan visto.

2 Analizar y evaluar algoritmos de aprendizaje supervisado y la precisión del modelo.

2.1 Analizar varios algoritmos de aprendizaje supervisado para comprender sus características y aplicaciones específicas.

Aprendizaje automático supervisado es una forma de aprendizaje automático en la que el algoritmo se entrena con datos etiquetados para realizar predicciones o tomar decisiones basadas en las en-

tradas de datos. En el aprendizaje supervisado, el algoritmo aprende una relación entre los datos de entrada y de salida. Esta relación se aprende a partir de un conjunto de datos etiquetados, que consta de pares de datos de entrada y de salida. El algoritmo intenta aprender la relación entre los datos de entrada y de salida para poder realizar predicciones precisas sobre datos nuevos e invisibles.

El aprendizaje supervisado es aquel en el que el modelo se entrena con un conjunto de datos etiquetado. Un conjunto de datos etiquetado es aquel que tiene parámetros de entrada y de salida. En este tipo de aprendizaje, tanto el entrenamiento como la validación.

El conjunto de datos etiquetados que se utiliza en el aprendizaje supervisado consta de características de entrada y etiquetas de salida correspondientes. Las características de entrada son los atributos o características de los datos que se utilizan para hacer predicciones, mientras que las etiquetas de salida son los resultados o metas deseados que el algoritmo intenta predecir.

Ejemplo 1: un conjunto de datos meteorológicos que sirve para predecir la velocidad del viento en función de diferentes parámetros. Datos de Entrada: punto de rocío, temperatura, presión, humedad relativa, dirección del viento Salida: velocidad del viento

Ejemplo 2: un conjunto de datos de una tienda de compras que es útil para predecir si un cliente comprará un producto en particular en consideración o no en función de su género, edad y salario. Datos de Entrada: Género, edad, salario Salida: Comprado, es decir, 0 o 1; 1 significa que el cliente sí lo comprará y 0 significa que el cliente no lo comprará.

Entrenamiento del sistema: durante el entrenamiento del modelo, los datos se suelen dividir en una proporción de 80:20, es decir, el 80 % se utiliza como datos de entrenamiento y el resto como datos de prueba. En el caso de los datos de entrenamiento, introducimos datos de entrada y de salida para el 80 % de los datos. El modelo aprende solo de los datos de entrenamiento. Utilizamos diferentes algoritmos de aprendizaje automático (que analizaremos en detalle en los próximos artículos) para crear nuestro modelo. Aprender significa que el modelo creará su propia lógica. Una vez que el modelo esté listo, será bueno probarlo. En el momento de la prueba, la entrada se introduce a partir del 20 % restante de datos que el modelo nunca ha visto antes, el modelo predecirá algún valor y lo compararemos con el resultado real y calcularemos la precisión.

Tipos de algoritmos de aprendizaje supervisado

El aprendizaje supervisado se divide generalmente en dos categorías principales: regresión y clasificación. En la regresión, el algoritmo aprende a predecir un valor de salida continuo, como el precio de una casa o la temperatura de una ciudad. En la clasificación, el algoritmo aprende a predecir una variable de salida categórica o una etiqueta de clase, como si es probable o no que un cliente compre un producto.

Una de las principales ventajas del aprendizaje supervisado es que permite la creación de modelos complejos que pueden realizar predicciones precisas a partir de nuevos datos. Sin embargo, el aprendizaje supervisado requiere grandes cantidades de datos de entrenamiento etiquetados para ser eficaz. Además, la calidad y la representatividad de los datos de entrenamiento pueden tener un impacto significativo en la precisión del modelo.

Regresión

La regresión es una técnica de aprendizaje supervisado que se utiliza para predecir valores numéricos continuos en función de las características de entrada. Su objetivo es establecer una relación funcional entre las variables independientes y una variable dependiente, como predecir los precios

de las viviendas en función de características como el tamaño, el número de habitaciones y la ubicación. El objetivo es minimizar la diferencia entre los valores previstos y los reales mediante algoritmos como la regresión lineal, los árboles de decisión o las redes neuronales, lo que garantiza que el modelo capture los patrones subyacentes en los datos.

Clasificación

La clasificación es un tipo de aprendizaje supervisado que clasifica los datos de entrada en etiquetas predefinidas. Implica entrenar un modelo con ejemplos etiquetados para aprender patrones entre las características de entrada y las clases de salida. En la clasificación, la variable de destino es un valor categórico. Por ejemplo, clasificar los correos electrónicos como spam o no. El objetivo del modelo es generalizar este aprendizaje para hacer predicciones precisas sobre datos nuevos e inéditos. Algoritmos como árboles de decisión, máquinas de vectores de soporte y redes neuronales se utilizan comúnmente para tareas de clasificación.

NOTA: Existen algoritmos de aprendizaje automático supervisado comunes que pueden usarse tanto para tareas de regresión como de clasificación.

Algoritmo de aprendizaje automático supervisado

El aprendizaje supervisado se puede dividir en varios tipos diferentes, cada uno con sus propias características y aplicaciones. Estos son algunos de los tipos más comunes de algoritmos de aprendizaje supervisado:

- Regresión lineal: la regresión lineal es un tipo de algoritmo de regresión que se utiliza para predecir un valor de salida continuo. Es uno de los algoritmos más simples y más utilizados en el aprendizaje supervisado. En la regresión lineal, el algoritmo intenta encontrar una relación lineal entre las características de entrada y el valor de salida. El valor de salida se predice en función de la suma ponderada de las características de entrada.
- Regresión logística: la regresión logística es un tipo de algoritmo de clasificación que se utiliza para predecir una variable de salida binaria. Se utiliza comúnmente en aplicaciones de aprendizaje automático donde la variable de salida es verdadera o falsa, como en la detección de fraudes o el filtrado de correo no deseado. En la regresión logística, el algoritmo intenta encontrar una relación lineal entre las características de entrada y la variable de salida. Luego, la variable de salida se transforma utilizando una función logística para producir un valor de probabilidad entre 0 y 1.
- Árboles de decisión (Decision Trees): un árbol de decisión es una estructura similar a un árbol que se utiliza para modelar decisiones y sus posibles consecuencias. Cada nodo interno del árbol representa una decisión, mientras que cada nodo de hoja representa un posible resultado. Los árboles de decisión se pueden utilizar para modelar relaciones complejas entre características de entrada y variables de salida. Un árbol de decisión es un tipo de algoritmo que se utiliza tanto para tareas de clasificación como de regresión.
  - Regresión de árboles de decisión (Decision Trees Regression): los árboles de decisión se pueden utilizar para tareas de regresión al predecir el valor vinculado con un nodo hoja.
  - Clasificación de árboles de decisión (Decision Trees Classification): Random Forest es un algoritmo de aprendizaje automático que utiliza múltiples árboles de decisión para mejorar la clasificación y evitar el sobreajuste.
- Bosques aleatorios (Random Forests): los bosques aleatorios se componen de varios árboles de decisión que trabajan juntos para hacer predicciones. Cada árbol del bosque se entrena

con un subconjunto diferente de las características y los datos de entrada. La predicción final se realiza agregando las predicciones de todos los árboles del bosque. Los bosques aleatorios son una técnica de aprendizaje conjunto que se utiliza tanto para tareas de clasificación como de regresión.

  - Regresión de bosque aleatorio (Random Forest Regression): combina múltiples árboles de decisión para reducir el sobreajuste y mejorar la precisión de la predicción.
  - Clasificador de bosque aleatorio (Random Forest Classifier): combina varios árboles de decisión para mejorar la precisión de la clasificación y minimizar el sobreajuste.

- Máquina de vectores de soporte (Support Vector Machine(SVM)) : el algoritmo SVM crea un hiperplano para segregar el espacio n-dimensional en clases e identificar la categoría correcta de nuevos puntos de datos. Los casos extremos que ayudan a crear el hiperplano se denominan vectores de soporte, de ahí el nombre de máquina de vectores de soporte. Una máquina de vectores de soporte es un tipo de algoritmo que se utiliza tanto para tareas de clasificación como de regresión.

  - Regresión de vectores de soporte (Support Vector Regression): es una extensión de las máquinas de vectores de soporte (SVM) que se utiliza para predecir valores continuos.
  - Clasificador de vectores de soporte (Support Vector Classifier): tiene como objetivo encontrar el mejor hiperplano que maximice el margen entre puntos de datos de diferentes clases.

- Vecinos más cercanos (K-Nearest Neighbors(KNN)): KNN funciona buscando los ejemplos de entrenamiento k más cercanos a una entrada dada y luego predice la clase o el valor en función de la clase mayoritaria o el valor promedio de estos vecinos. El rendimiento de KNN puede verse influenciado por la elección de k y la métrica de distancia utilizada para medir la proximidad. Sin embargo, es intuitivo pero puede ser sensible a los datos ruidosos y requiere una selección cuidadosa de k para obtener resultados óptimos. Un vecino más cercano (KNN) es un tipo de algoritmo que se utiliza tanto para tareas de clasificación como de regresión.

  - Regresión de los k vecinos más cercanos (K-Nearest Neighbors Regression): predice valores continuos promediando las salidas de los k vecinos más cercanos.
  - Clasificación de K vecinos más cercanos (K-Nearest Neighbors Classification): los puntos de datos se clasifican según la clase mayoritaria de sus k vecinos más cercanos.

- Gradient Boosting: el Gradient Boosting combina modelos débiles, como árboles de decisión, para crear un modelo sólido. Construye iterativamente nuevos modelos que corrigen errores cometidos por los anteriores. Cada nuevo modelo se entrena para minimizar los errores residuales, lo que da como resultado un predictor poderoso capaz de manejar relaciones de datos complejas. El Gradient Boosting es un tipo de algoritmo que se utiliza tanto para tareas de clasificación como de regresión.

  - Regresión de aumento de gradiente (Gradient Boosting Regression): crea un conjunto de estudiantes débiles para mejorar la precisión de la predicción a través del entrenamiento iterativo.
  - Clasificación de aumento de gradiente (Gradient Boosting Classification): crea un grupo de clasificadores para mejorar continuamente la precisión de las predicciones a través de iteraciones

2.2 Evaluar los conceptos de sobreajuste y subajuste en estos modelos, incluyendo una explicación detallada del equilibrio entre sesgo y varianza.

Underfitting and Overfitting

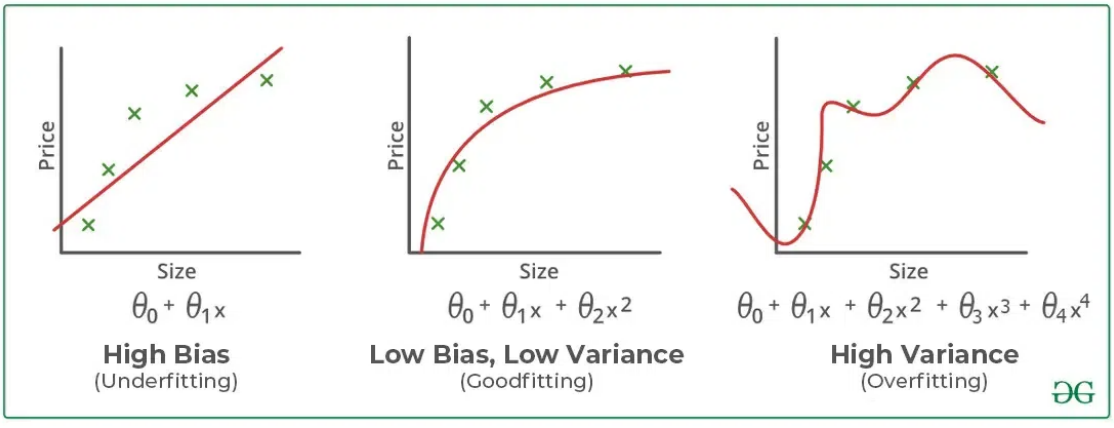
Cuando hablamos del modelo de aprendizaje automático, en realidad hablamos de su rendimiento y su precisión, lo que se conoce como errores de predicción. Consideremos que estamos diseñando un modelo de aprendizaje automático. Se dice que un modelo es un buen modelo de aprendizaje automático si generaliza cualquier dato de entrada nuevo del dominio del problema de forma adecuada. Esto nos ayuda a hacer predicciones sobre datos futuros que el modelo de datos nunca ha visto. Ahora, supongamos que queremos comprobar qué tan bien nuestro modelo de aprendizaje automático aprende y generaliza a los nuevos datos. Para ello, tenemos el sobreajuste y el subajuste, que son los principales responsables del bajo rendimiento de los algoritmos de aprendizaje automático .

Bias and Variance in Machine Learning

- Sesgo (Bias): el sesgo se refiere al error debido a suposiciones demasiado simplistas en el algoritmo de aprendizaje. Estas suposiciones hacen que el modelo sea más fácil de comprender y aprender, pero es posible que no capturen las complejidades subyacentes de los datos. Es el error debido a la incapacidad del modelo para representar la verdadera relación entre la entrada y la salida con precisión. Cuando un modelo tiene un rendimiento deficiente tanto en los datos de entrenamiento como en los de prueba, significa que hay un sesgo alto debido al modelo simple, lo que indica un ajuste insuficiente.
- Varianza (Variance): la varianza, por otro lado, es el error debido a la sensibilidad del modelo a las fluctuaciones en los datos de entrenamiento. Es la variabilidad de las predicciones del modelo para diferentes instancias de datos de entrenamiento. Una alta varianza ocurre cuando un modelo aprende el ruido y las fluctuaciones aleatorias de los datos de entrenamiento en lugar del patrón subyacente. Como resultado, el modelo funciona bien en los datos de entrenamiento, pero mal en los datos de prueba, lo que indica un **sobreajuste**.

[2]:
```
display.Image('./images/ajuste.png')
```

[2]:



El subajuste en el aprendizaje automático

Se dice que un modelo estadístico o un algoritmo de aprendizaje automático tiene un ajuste insuficiente cuando un modelo es demasiado simple para capturar las complejidades de los datos. Representa la incapacidad del modelo para aprender los datos de entrenamiento de manera efectiva, lo que da como resultado un rendimiento deficiente tanto en los datos de entrenamiento como en los de prueba. En términos simples, los modelos con ajuste insuficiente son inexactos, especialmente cuando se aplican a ejemplos nuevos e inéditos. Esto ocurre principalmente cuando utilizamos un modelo muy simple con suposiciones demasiado simplificadas. Para abordar el problema del ajuste insuficiente del modelo, necesitamos utilizar modelos más complejos, con una representación de características mejorada y menos regularización.

Nota: El modelo de subajuste tiene alto sesgo y baja varianza.

Razones para el subajuste

- El modelo es demasiado simple, por lo que es posible que no sea capaz de representar las complejidades de los datos.
- Las características de entrada que se utilizan para entrenar el modelo no son representaciones adecuadas de los factores subyacentes que influyen en la variable objetivo.
- El tamaño del conjunto de datos de entrenamiento utilizado no es suficiente.
- Se utiliza una regularización excesiva para evitar el sobreajuste, que impide que el modelo capture bien los datos.
- Las características no están escaladas.

Técnicas para reducir el desajuste

- Aumentar la complejidad del modelo.
- Aumente el número de funciones mediante la realización de ingeniería de funciones .
- Eliminar el ruido de los datos.
- Aumente el número de épocas o aumente la duración del entrenamiento para obtener mejores resultados.

Sobreajuste en el aprendizaje automático

Se dice que un modelo estadístico está sobreajustado cuando el modelo no hace predicciones precisas sobre los datos de prueba. Cuando un modelo se entrena con tantos datos, comienza a aprender del ruido y de las entradas de datos inexactas en nuestro conjunto de datos. Y cuando se prueba con datos de prueba, los resultados son de alta varianza. Entonces, el modelo no categoriza los datos correctamente, debido a demasiados detalles y ruido. Las causas del sobreajuste son los métodos no paramétricos y no lineales porque estos tipos de algoritmos de aprendizaje automático tienen más libertad para construir el modelo en función del conjunto de datos y, por lo tanto, pueden construir modelos realmente poco realistas. Una solución para evitar el sobreajuste es usar un algoritmo lineal si tenemos datos lineales o usar parámetros como la profundidad máxima si estamos usando árboles de decisión.

En pocas palabras, el sobreajuste es un problema en el que la evaluación de los algoritmos de aprendizaje automático en datos de entrenamiento es diferente a la de los datos no vistos.

Razones para el sobreajuste:

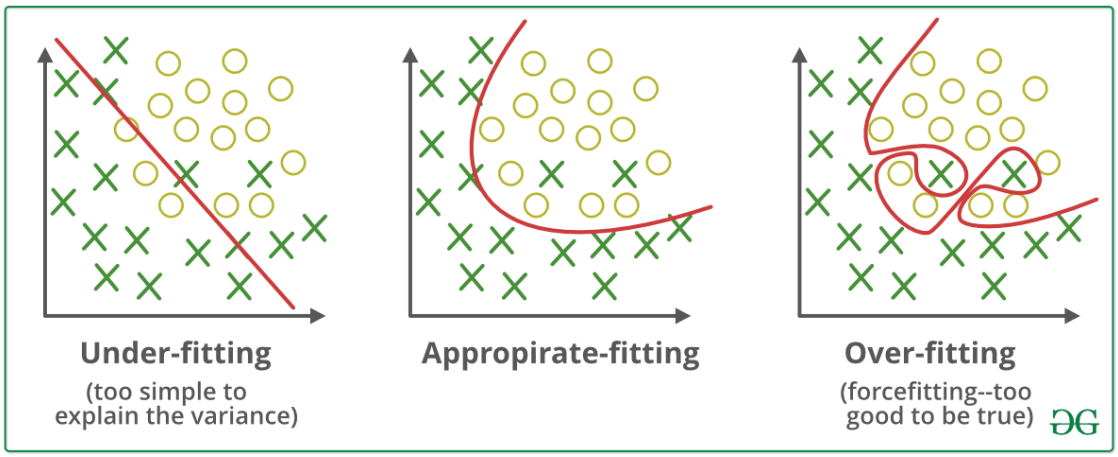
- Alta varianza y bajo sesgo.
- El modelo es demasiado complejo.
- El tamaño de los datos de entrenamiento.

Técnicas para reducir el sobreajuste

- Mejorar la calidad de los datos de entrenamiento reduce el sobreajuste al centrarse en patrones significativos y mitigar el riesgo de ajustar el ruido o las características irrelevantes.
- Aumentar los datos de entrenamiento puede mejorar la capacidad del modelo para generalizarse a datos no vistos y reducir la probabilidad de sobreajuste.
- Reducir la complejidad del modelo.
- Parada temprana durante la fase de entrenamiento (vigile la pérdida durante el período de entrenamiento y tan pronto como la pérdida comience a aumentar, detenga el entrenamiento).
- Regularización de crestas y regularización de lazo .
- Utilice la deserción en redes neuronales para abordar el sobreajuste.

[3]:
```
display.Image('./images/ajuste2.png')
```

[3]:



Buen ajuste en un modelo estadístico

Idealmente, se dice que el modelo tiene un buen ajuste a los datos cuando realiza predicciones con un error de cero. Esta situación se puede lograr en un punto entre el sobreajuste y el subajuste. Para entenderlo, tendremos que observar el rendimiento de nuestro modelo con el paso del tiempo, mientras aprende del conjunto de datos de entrenamiento.

Con el paso del tiempo, nuestro modelo seguirá aprendiendo y, por lo tanto, el error del modelo en los datos de entrenamiento y prueba seguirá disminuyendo. Si aprende durante demasiado tiempo, el modelo será más propenso a sobreajustarse debido a la presencia de ruido y detalles menos útiles. Por lo tanto, el rendimiento de nuestro modelo disminuirá. Para obtener un buen ajuste, nos detendremos en un punto justo antes de que el error comience a aumentar. En este punto, se dice que el modelo tiene buenas habilidades en los conjuntos de datos de entrenamiento, así como en nuestro conjunto de datos de prueba no visto.

Cross-Validation

La validación cruzada o cross-validation es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la predicción y se

quiere estimar la precisión de un modelo que se llevará a cabo a la práctica. Es una técnica muy utilizada en proyectos de inteligencia artificial para validar modelos generados.

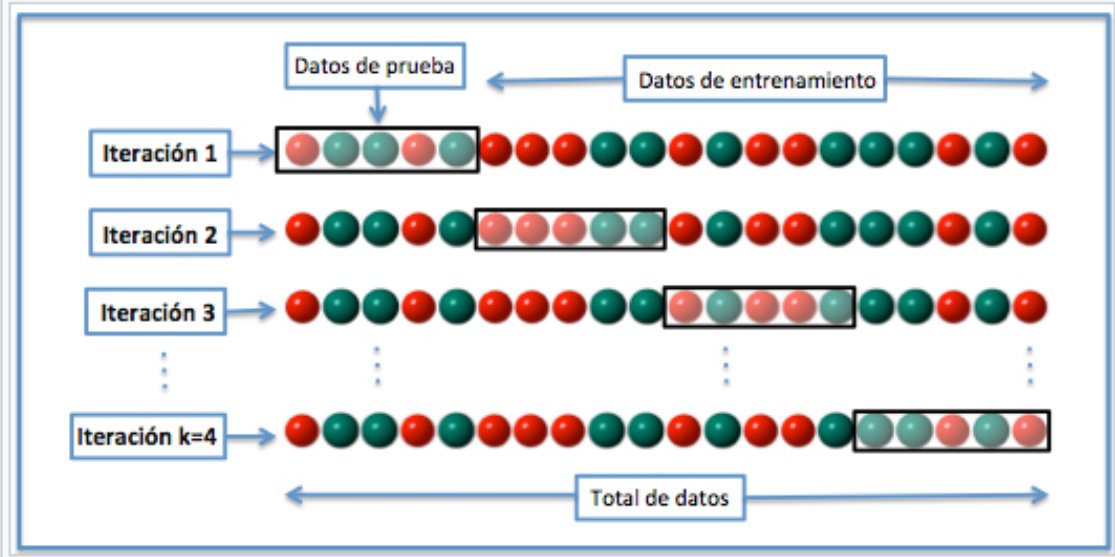
Objetivo

Suponemos que tenemos un modelo con uno o más parámetros de ajuste desconocidos y unos datos de entrenamiento que queremos analizar. El proceso de ajuste optimiza los parámetros del modelo para que éste se ajuste a los datos de entrenamiento tan bien como pueda. Si tomamos una muestra independiente como dato de prueba (validación), del mismo grupo que los datos de entrenamiento, normalmente el modelo no se ajustará a los datos de prueba igual de bien que a los datos de entrenamiento. Esto se denomina sobreajuste y acostumbra a pasar cuando el tamaño de los datos de entrenamiento es pequeño o cuando el número de parámetros del modelo es grande. La validación cruzada es una manera de predecir el ajuste de un modelo a un hipotético conjunto de datos de prueba cuando no disponemos del conjunto explícito de datos de prueba.

- Validación cruzada de K iteraciones o K-fold cross-validation los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado. Este método es muy preciso puesto que evaluamos a partir de K combinaciones de datos de entrenamiento y de prueba, pero aun así tiene una desventaja, y es que, a diferencia del método de retención, es lento desde el punto de vista computacional. En la práctica, la elección del número de iteraciones depende de la medida del conjunto de datos. Lo más común es utilizar la validación cruzada de 10 iteraciones (10-fold cross-validation).

[4]:
```
display.Image('./images/k-fold.png')
```

[4]:



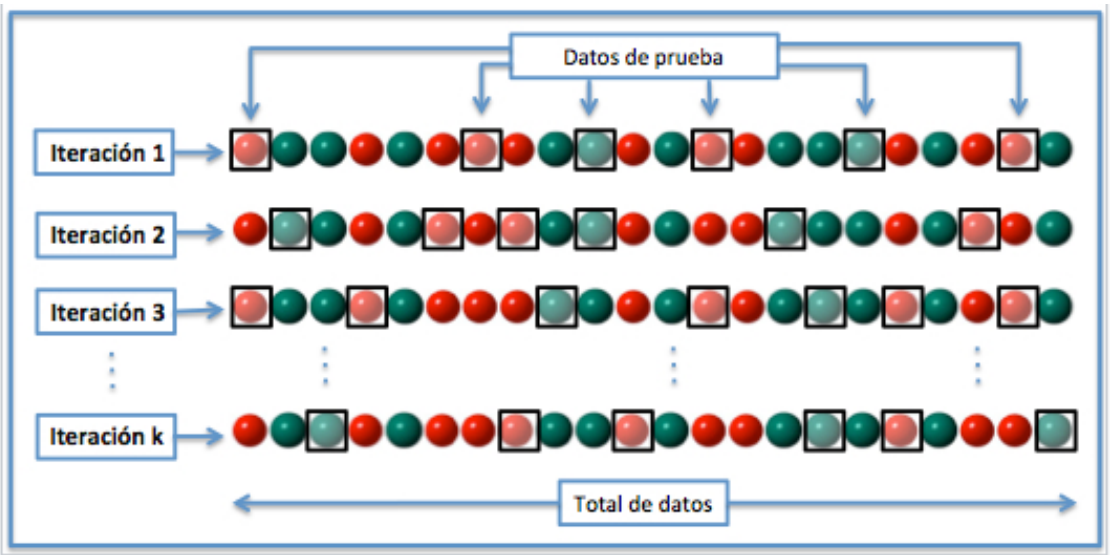
Validación cruzada de K iteraciones con *K*=4.

- Validación cruzada aleatoria (Random permutations cross-validation): Este método consiste

al dividir aleatoriamente el conjunto de datos de entrenamiento y el conjunto de datos de prueba. Para cada división la función de aproximación se ajusta a partir de los datos de entrenamiento y calcula los valores de salida para el conjunto de datos de prueba. El resultado final se corresponde a la media aritmética de los valores obtenidos para las diferentes divisiones. La ventaja de este método es que la división de datos entrenamiento-prueba no depende del número de iteraciones. Pero, en cambio, con este método hay algunas muestras que quedan sin evaluar y otras que se evalúan más de una vez, es decir, los subconjuntos de prueba y entrenamiento se pueden solapar.

[5]:
```
display.Image('./images/aleatoria.png')
```

[5]:

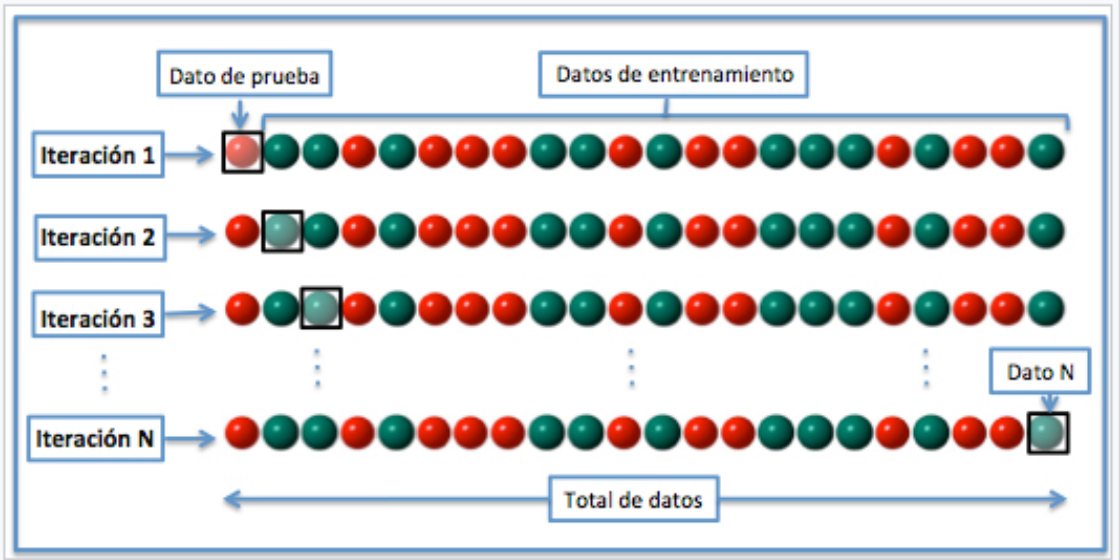


Validación cruzada aleatória con *k* iteraciones.

- Validación cruzada dejando uno fuera o Leave-one-out cross-validation (LOOCV) implica separar los datos de forma que para cada iteración tengamos una sola muestra para los datos de prueba y todo el resto conformando los datos de entrenamiento. La evaluación viene dada por el error, y en este tipo de validación cruzada el error es muy bajo, pero en cambio, a nivel computacional es muy costoso, puesto que se tienen que realizar un elevado número de iteraciones, tantas como N muestras tengamos y para cada una analizar los datos tanto de entrenamiento como de prueba

[6]:
```
display.Image('./images/LOOCV.png')
```

[6]:



Validación cruzada dejando uno fuera *(LOOCV)*.

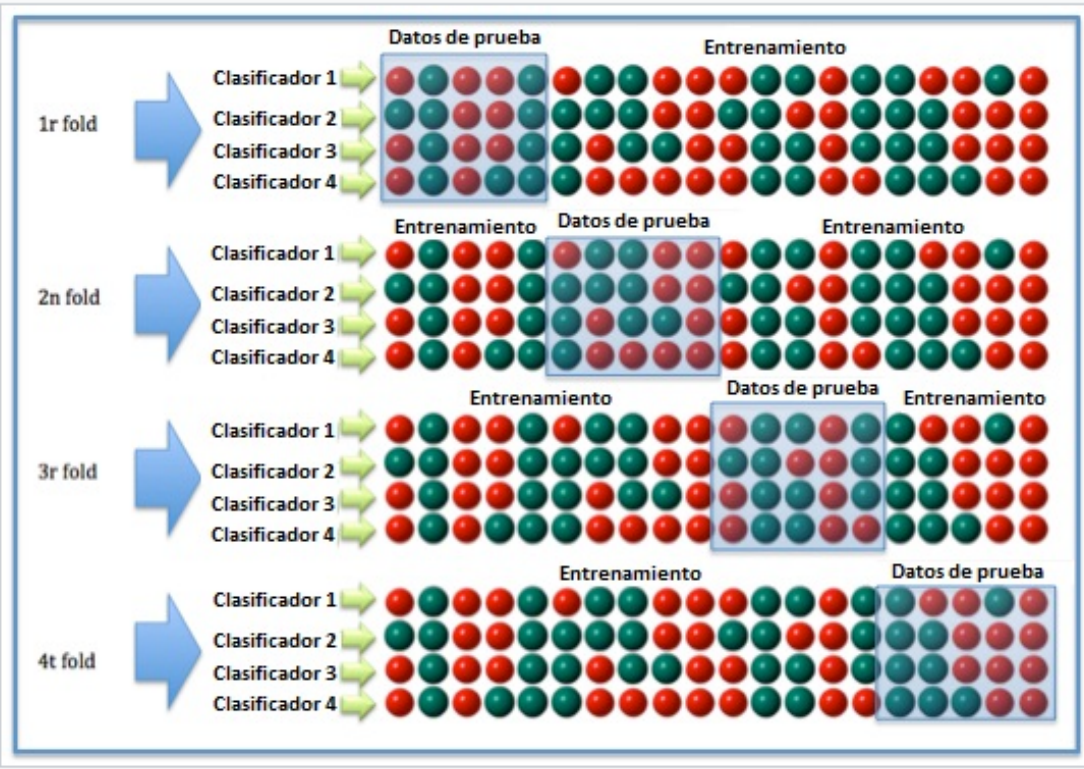
Ejemplos de validación cruzada

La validación cruzada se puede utilizar para comparar los resultados de diferentes procedimientos de clasificación predictiva. Por ejemplo, supongamos que tenemos un detector que nos determina si una cara pertenece a una mujer o a un hombre y consideramos que han sido utilizados dos métodos diferentes, por ejemplo, máquinas de vectores de soporte (SVM) y K-vecinos más cercanos (Knn), ya que ambos nos permiten clasificar las imágenes. Con la validación cruzada podríamos comparar los dos procedimientos y determinar cuál de los dos es el más preciso. Esta información nos la proporciona la tasa de error que obtenemos al aplicar la validación cruzada por cada uno de los métodos planteados.

La validación cruzada de "k" iteraciones (k-fold cross validation) nos permite evaluar también modelos en los que se utilizan varios clasificadores. Continuando con el ejemplo anterior, si tenemos un detector que nos determina si en una imagen aparece un hombre o una mujer, y éste utiliza cuatro clasificadores binarios para detectarlo, también podemos utilizar la validación cruzada para evaluar su precisión. Si tenemos un total de 20 datos (imágenes), y utilizamos el método 4-fold cross validation, se llevarán a cabo cuatro iteraciones, y en cada una se utilizarán unos datos de entrenamiento diferentes, que serán analizadas por cuatro clasificadores, que posteriormente evaluarán los datos de prueba. De este modo por cada muestra obtendremos cuatro resultados, y si hacemos la media entre los resultados de cada clasificador y entre las cuatro iteraciones realizadas, obtendremos el valor resultante final.

[7]:
```
display.Image('./images/ejemplo_val.png')
```

[7]:



k-fold cross validation, con k=4 y con 4 clasificadores.

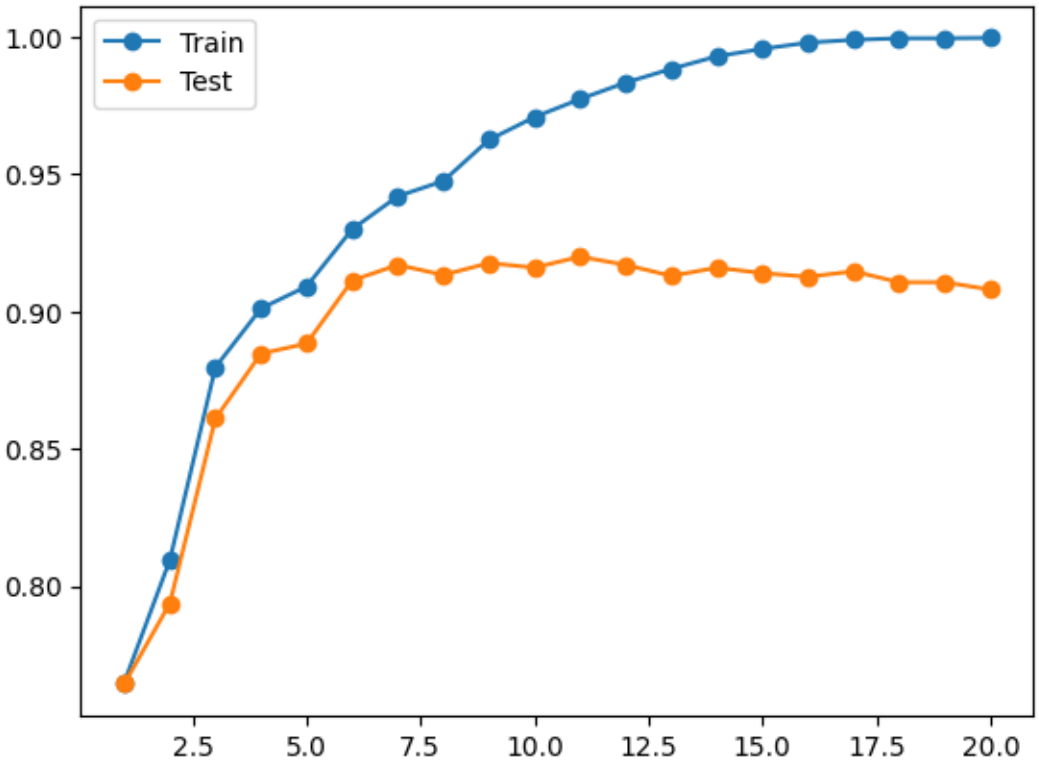
- **Ejemplo [1]**

[8]:
```
# pip install scikit-learn
```

[9]:
```
# evaluate decision tree performance on train and test sets with different tree␣
 ↪depths
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot
# create dataset
X, y = make_classification(n_samples=10000, n_features=20, n_informative=5,␣
 ↪n_redundant=15, random_state=1)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# define lists to collect scores
train_scores, test_scores = list(), list()
# define the tree depths to evaluate
values = [i for i in range(1, 21)]
# evaluate a decision tree for each depth
```

```
for i in values:
        # configure the model
        model = DecisionTreeClassifier(max_depth=i)
        # fit model on the training dataset
        model.fit(X_train, y_train)
        # evaluate on the train dataset
        train_yhat = model.predict(X_train)
        train_acc = accuracy_score(y_train, train_yhat)
        train_scores.append(train_acc)
        # evaluate on the test dataset
        test_yhat = model.predict(X_test)
        test_acc = accuracy_score(y_test, test_yhat)
        test_scores.append(test_acc)
        # summarize progress
        print('>%d, train: %.3f, test: %.3f' % (i, train_acc, test_acc))
# plot of train and test scores vs tree depth
pyplot.plot(values, train_scores, '-o', label='Train')
pyplot.plot(values, test_scores, '-o', label='Test')
pyplot.legend()
pyplot.show()
```

```
>1, train: 0.765, test: 0.764
>2, train: 0.809, test: 0.794
>3, train: 0.880, test: 0.861
>4, train: 0.901, test: 0.885
>5, train: 0.909, test: 0.888
>6, train: 0.930, test: 0.911
>7, train: 0.942, test: 0.917
>8, train: 0.948, test: 0.913
>9, train: 0.963, test: 0.918
>10, train: 0.971, test: 0.916
>11, train: 0.977, test: 0.920
>12, train: 0.983, test: 0.917
>13, train: 0.988, test: 0.913
>14, train: 0.993, test: 0.916
>15, train: 0.996, test: 0.914
>16, train: 0.998, test: 0.913
>17, train: 0.999, test: 0.915
>18, train: 1.000, test: 0.911
>19, train: 1.000, test: 0.911
>20, train: 1.000, test: 0.908
```



También se crea una figura que muestra gráficos de líneas de la precisión del modelo en los conjuntos de entrenamiento y de prueba con diferentes profundidades de árbol.

El gráfico muestra claramente que aumentar la profundidad del árbol en las primeras etapas resulta en una mejora correspondiente tanto en los conjuntos de entrenamiento como de prueba.

Esto continúa hasta una profundidad de aproximadamente 10 niveles, tras la cual se observa que el modelo sobreajusta el conjunto de datos de entrenamiento, a costa de un peor rendimiento en el conjunto de datos de reserva.

Este análisis es interesante. Muestra por qué el modelo tiene un peor rendimiento en el conjunto de pruebas de retención cuando "max_depth" se establece en valores altos.

Podemos elegir fácilmente un valor de "max_depth" mediante una búsqueda en cuadrícula sin analizar por qué algunos valores resultan en un mejor rendimiento y otros en un peor rendimiento.

- **Ejemplo [2]**

El Conjunto de Datos de Vivienda de California, recopilado por Pace y Barry (1997), contiene datos de la Encuesta Nacional de 1990. Censo sobre la vivienda en California. Incluye 20.640 observaciones con características como ubicación, antigüedad de la vivienda y población. La variable objetivo es el valor medio de la vivienda en los distritos de California.

[10]:
```
from sklearn.datasets import fetch_california_housing
import pandas as pd
```

```python
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import numpy as np

# caragmos el dataset
data = fetch_california_housing()

# dividimos los datos de variable depedendiente e independiente:
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

def calculate_kFold(k):
    # realizamos el K-fold para 5 pruebas
    kf = KFold(n_splits=k, shuffle=True, random_state=42)

    # creamos el modelo
    model = LinearRegression()

     # sacamos la precisión del modelo para cada prueba
    scores = cross_val_score(model, X, y, cv=kf, scoring='r2')


    # calculamos la media de las pruebas:
    average_r2 = np.mean(scores)

    print(f"R² Score for each fold: {[round(score, 4) for score in scores]}")
    print(f"Average R² across {k} folds: {average_r2:.2f}")
```

[11]:
```python
calculate_kFold(5)
```

```
R² Score for each fold: [np.float64(0.5758), np.float64(0.6137),
np.float64(0.6086), np.float64(0.6213), np.float64(0.5875)]
Average R² across 5 folds: 0.60
```

[12]:
```python
calculate_kFold(3)
```

```
R² Score for each fold: [np.float64(0.5967), np.float64(0.6132),
np.float64(0.6007)]
Average R² across 3 folds: 0.60
```

[13]:
```python
calculate_kFold(20)
```

```
R² Score for each fold: [np.float64(0.6077), np.float64(0.5527),
np.float64(0.6109), np.float64(0.5277), np.float64(0.6519), np.float64(0.6154),
np.float64(0.609), np.float64(0.5779), np.float64(0.6292), np.float64(0.6019),
np.float64(0.6217), np.float64(0.5847), np.float64(0.6265), np.float64(0.6156),
np.float64(0.6511), np.float64(0.6289), np.float64(0.5424), np.float64(0.6139),
np.float64(0.5863), np.float64(0.5999)]
```

```
Average R² across 20 folds: 0.60
```

Este es el resultado del código del paso anterior. Muestra las puntuaciones R^2 de cada muestra y la puntuación R^2 media.

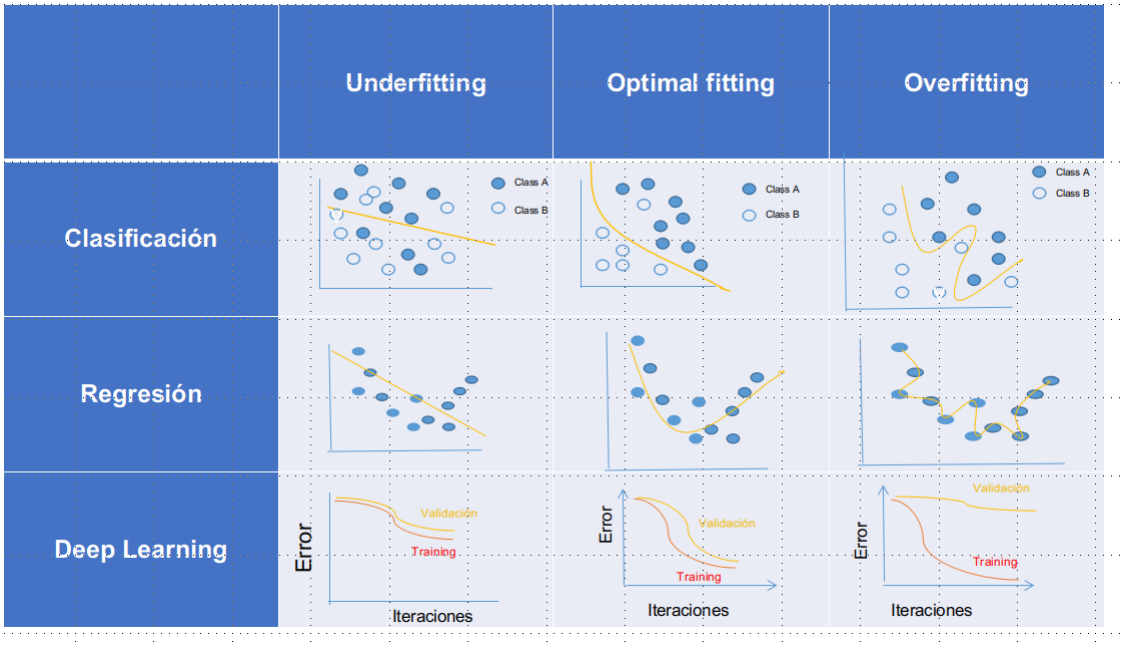
En la validación cruzada de K pliegues, "K" representa el número de grupos en que se divide el conjunto de datos. Este número determina cuántas rondas de pruebas se someten al modelo, asegurándose de que cada segmento se utiliza como conjunto de pruebas una vez.

He aquí una heurística:

- K = 2 ó 3: Estas opciones pueden ser beneficiosas cuando los recursos computacionales son limitados o cuando se necesita una evaluación más rápida. Reducen el número de ciclos de entrenamiento, con lo que ahorran tiempo y potencia de cálculo, al tiempo que proporcionan una estimación razonable del rendimiento del modelo.
- K = 5 ó 10: Elegir K = 5 o K = 10 son opciones populares porque proporcionan un buen equilibrio entre la eficiencia computacional y la estimación del rendimiento del modelo.
- K = 20: Utilizar un valor mayor de K puede proporcionar una evaluación más detallada del rendimiento. Sin embargo, aumenta la carga computacional y puede dar lugar a una mayor varianza si los subconjuntos son demasiado pequeños.

[14]:
```
display.Image('./images/entrenamientos.png')
```

[14]:



Creado por:

Isabel Maniega