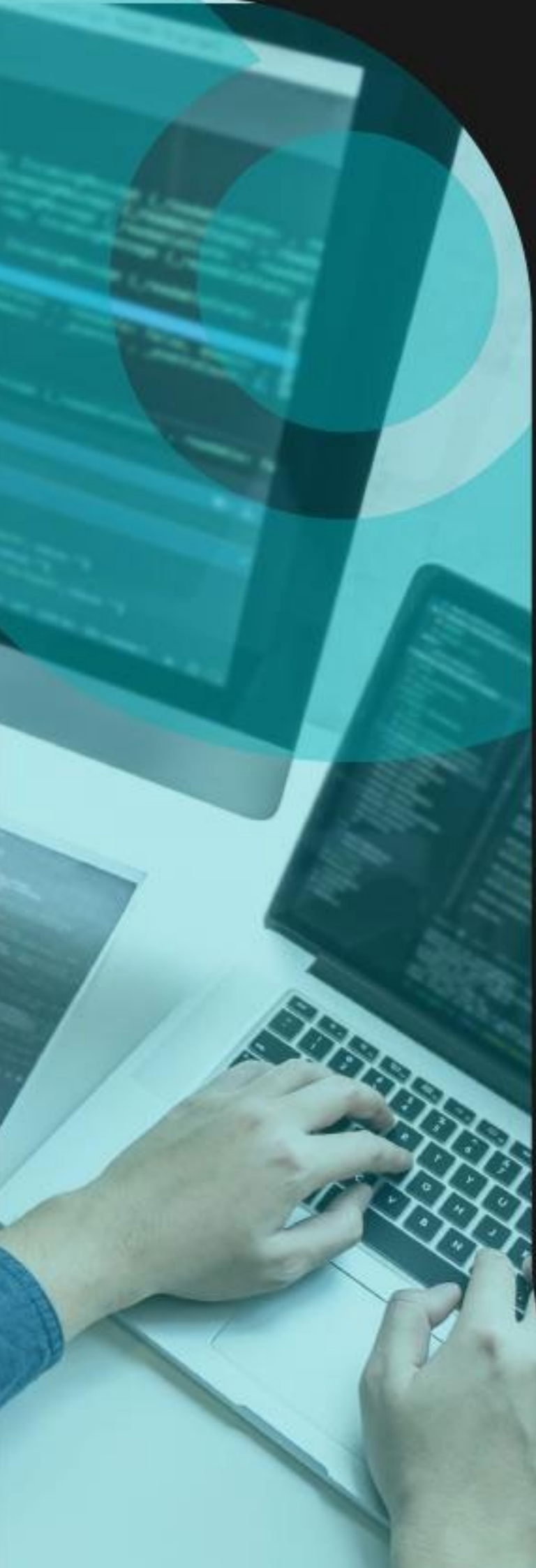


AWAKELAB

BASECAMP

Ciencia de Datos

Módulo: Aprendizaje de Máquina Supervisado

Aprendizaje Esperado

---

9. Elabora un modelo predictivo aplicando técnicas de Gradient Boosting para resolver un problema de clasificación utilizando lenguaje Python.

El origen del Boosting

La idea de impulsar surgió de la idea de si un alumno débil puede modificarse para mejorar.

Michael Kearns articuló el objetivo como el *"problema del boosting (impulso) de hipótesis es …un algoritmo eficiente para convertir hipótesis relativamente pobres en hipótesis muy buenas"* — Thoughts on Hypothesis Boosting [PDF], 1988

El marco estadístico presenta el impulso como un problema de optimización numérica en el que el objetivo es minimizar la pérdida del modelo mediante la adición de alumnos débiles mediante un procedimiento similar al descenso de gradiente.

Esta clase de algoritmos se describió como un modelo aditivo por etapas. Esto se debe a que se agrega un nuevo alumno débil a la vez y los alumnos débiles existentes en el modelo se congelan y no se modifican.
Tenga en cuenta que esta estrategia por etapas es diferente de los enfoques por etapas que reajustan los términos ingresados previamente cuando se agregan otros nuevos.

La generalización permitió el uso de funciones de pérdida diferenciables arbitrarias, expandiendo la técnica más allá de los problemas de clasificación binaria para admitir la regresión, la clasificación multiclase y más.

Cómo funciona el Gradient Boosting

El aumento de gradiente involucra tres elementos:

1. Una función de pérdida a optimizar.
2. Un aprendiz débil para hacer predicciones.
3. Un modelo aditivo para agregar estudiantes débiles para minimizar la función de pérdida.

1. Función de pérdida

La función de pérdida utilizada depende del tipo de problema que se está resolviendo. Debe ser diferenciable, pero se admiten muchas funciones de pérdida estándar y puede definir las suyas propias.

Por ejemplo, la regresión puede usar un error cuadrático y la clasificación puede usar una pérdida logarítmica. Un beneficio del marco de aumento de gradiente es que no es necesario derivar un nuevo algoritmo de aumento para cada función de pérdida que se quiera usar, sino que es un marco lo suficientemente genérico como para que se pueda usar cualquier función de pérdida diferenciable.

2. Aprendiz débil

Los árboles de decisión se utilizan como el alumno débil en el aumento de gradiente. Específicamente, se utilizan árboles de regresión que arrojan valores reales para divisiones y cuya salida se puede sumar, lo que permite agregar salidas de modelos posteriores y "corregir" los residuos en las predicciones.

Los árboles se construyen de manera codiciosa, eligiendo los mejores puntos de división en función de las puntuaciones de pureza como Gini o para minimizar la pérdida. Inicialmente, como en el caso de AdaBoost, se usaban árboles de decisión muy cortos que solo tenían una única división, llamada muñón de decisión. Los árboles más grandes se pueden usar generalmente con niveles de 4 a 8.

Es común restringir a los alumnos débiles de formas específicas, como un número máximo de capas, nodos, divisiones o nodos hoja. Esto es para

garantizar que los alumnos sigan siendo débiles, pero que aún puedan construirse de manera codiciosa.

3. Modelo aditivo

Los árboles se agregan de uno en uno y los árboles existentes en el modelo no se modifican. Se utiliza un procedimiento de descenso de gradiente para minimizar la pérdida al agregar árboles.

Tradicionalmente, el descenso de gradiente se usa para minimizar un conjunto de parámetros, como los coeficientes en una ecuación de regresión o los pesos en una red neuronal. Después de calcular el error o la pérdida, los pesos se actualizan para minimizar ese error.

En lugar de parámetros, tenemos submodelos de aprendizaje débiles o, más específicamente, árboles de decisión. Después de calcular la pérdida, para realizar el procedimiento de descenso de gradiente, debemos agregar un árbol al modelo que reduzca la pérdida (es decir, siga el gradiente). Hacemos esto parametrizando el árbol, luego modificamos los parámetros del árbol y nos movemos en la dirección correcta (reduciendo la pérdida residual). La salida del nuevo árbol se agrega luego a la salida de la secuencia de árboles existente en un esfuerzo por corregir o mejorar la salida final del modelo.

Se agrega una cantidad fija de árboles o el entrenamiento se detiene una vez que la pérdida alcanza un nivel aceptable o ya no mejora en un conjunto de datos de validación externa.

Mejoras en el aumento de gradiente básico

El aumento de gradiente es un algoritmo codicioso y puede sobreajerar un conjunto de datos de entrenamiento rápidamente. Puede beneficiarse de los métodos de regularización que penalizan varias partes del algoritmo y, en general, mejoran el rendimiento del algoritmo al reducir el sobreajuste.

En esta sección, veremos 4 mejoras en el aumento de gradiente básico:

1. Restricciones de árbol
2. Contracción
3. Muestreo aleatorio
4. Aprendizaje penalizado

1. Restricciones de árbol

Es importante que los alumnos débiles tengan habilidad pero sigan siendo débiles. Hay varias formas de restringir los árboles.
Una buena heurística general es que cuanto más restringida sea la creación de árboles, más árboles necesitará en el modelo, y al revés, donde menos árboles individuales restringidos, menos árboles serán necesarios.

A continuación se presentan algunas restricciones que se pueden imponer en la construcción de árboles de decisión:

- **Número de árboles** , por lo general, agregar más árboles al modelo puede ser muy lento para sobreajustar. El consejo es seguir agregando árboles hasta que no se observen más mejoras.
- **Profundidad del árbol** , los árboles más profundos son árboles más complejos y se prefieren árboles más cortos. Generalmente, se ven mejores resultados con 4-8 niveles.
- **Número de nodos o número de hojas** , como profundidad, esto puede restringir el tamaño del árbol, pero no está restringido a una estructura simétrica si se utilizan otras restricciones.
- **El número de observaciones por división** impone una restricción mínima sobre la cantidad de datos de entrenamiento en un nodo de entrenamiento antes de que se pueda considerar una división
- **La mejora mínima a la pérdida** es una restricción en la mejora de cualquier división agregada a un árbol.

2. Actualizaciones ponderadas

Las predicciones de cada árbol se suman secuencialmente.
La contribución de cada árbol a esta suma se puede ponderar para ralentizar el aprendizaje del algoritmo. Esta ponderación se denomina reducción o tasa de aprendizaje.
El efecto es que el aprendizaje se ralentiza, lo que a su vez requiere que se agreguen más árboles al modelo, lo que a su vez lleva más tiempo

entrenar, lo que proporciona un compromiso de configuración entre el número de árboles y la tasa de aprendizaje.

3. Impulso de gradiente estocástico

Una gran idea de los conjuntos de embolsado y el bosque aleatorio fue permitir que los árboles se crearán con avidez a partir de submuestras del conjunto de datos de entrenamiento.
Este mismo beneficio se puede utilizar para reducir la correlación entre los árboles en la secuencia en modelos de aumento de gradiente.
Esta variación de impulso se denomina aumento de gradiente estocástico.
En cada iteración, se extrae una submuestra de los datos de entrenamiento al azar (sin reemplazo) del conjunto de datos de entrenamiento completo. Luego se usa la submuestra seleccionada al azar, en lugar de la muestra completa, para ajustarse al alumno base.
— Aumento de gradiente estocástico [PDF], 1999

Algunas variantes de impulso estocástico que se pueden utilizar:

- Filas de submuestreo antes de crear cada árbol.
- Columnas de submuestra antes de crear cada árbol
- Columnas de submuestreo antes de considerar cada división.
-

En general, el submuestreo agresivo, como seleccionar solo el 50% de los datos, ha demostrado ser beneficioso.

4. Aumento de gradiente penalizado

Se pueden imponer restricciones adicionales a los árboles parametrizados además de su estructura. Los árboles de decisión clásicos como CART no se usan como aprendices débiles, sino que se usa una forma modificada llamada árbol de regresión que tiene valores numéricos en los nodos hoja (también llamados nodos terminales). Los valores en las hojas de los árboles pueden llamarse pesos en alguna literatura.

Como tal, los valores de peso de las hojas de los árboles se pueden regularizar utilizando funciones de regularización populares, como:

- L1 regularización de pesos.
- L2 regularización de pesos.

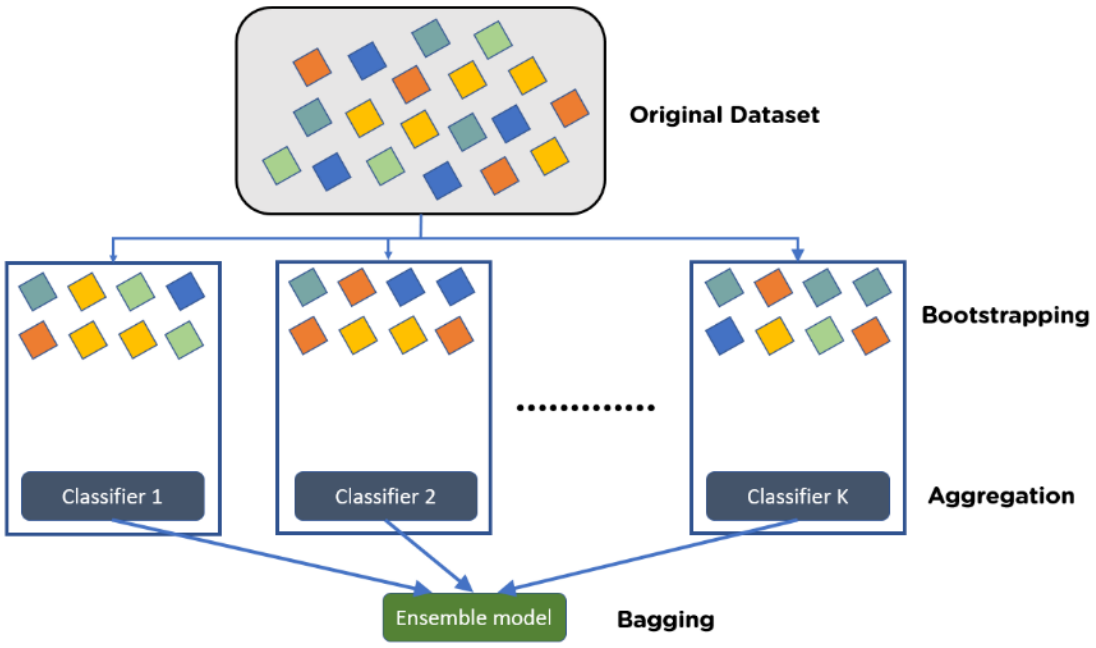
Bagging

Los árboles de decisión sufren de alta varianza, lo que significa que, si dividiéramos al azar los datos de entrenamiento en dos grupos y ajustamos un árbol de decisión a cada mitad, los resultados que obtendremos podrían ser radicalmente diferentes.

El método de *bagging* o *bootstrap aggregation* es una técnica de Machine Learning utilizada para reducir la varianza de un método de aprendizaje estadístico, usado muy frecuentemente con árboles de decisión.

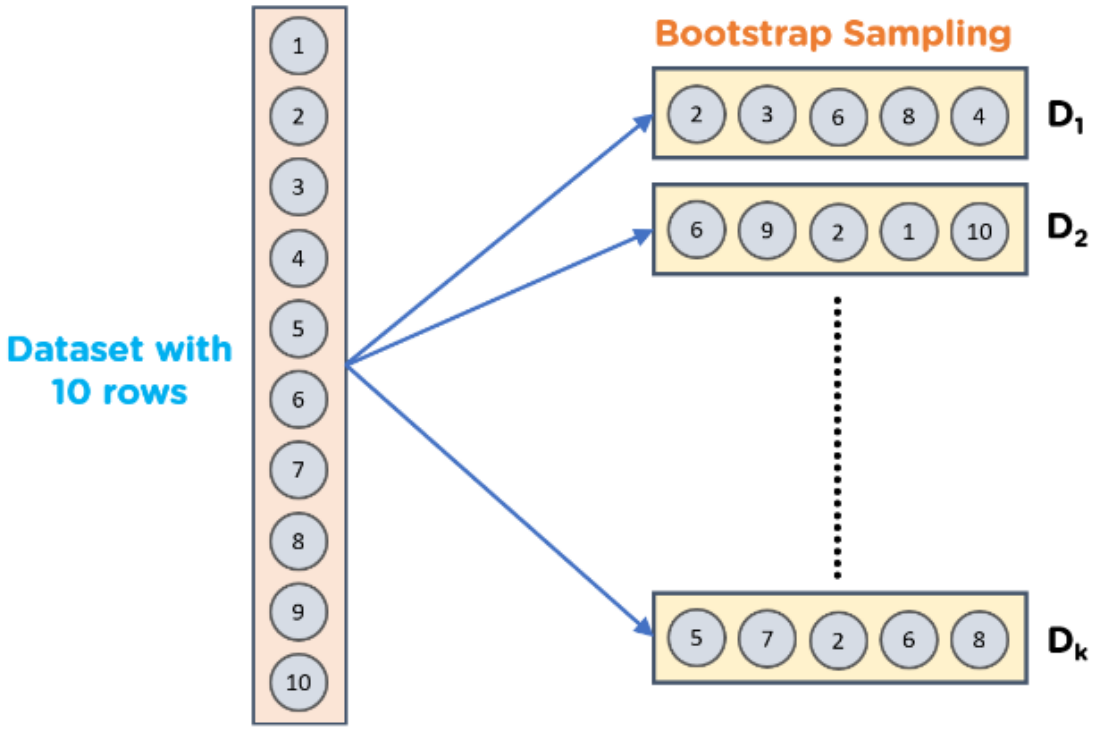
Procedimiento

1. Producir varios conjuntos de entrenamiento.
2. Crear para cada conjunto de entrenamiento, un modelo (en este caso, un árbol de decisión).
3. Combinar las predicciones de todos estos modelos obtenidos.



¿Qué es el bootstrapping?

Bootstrapping es el método de creación aleatoria de muestras de datos de una población con reemplazo para estimar un parámetro de población.



Ventajas del boosting en el aprendizaje automático

- El boosting minimiza el sobreajuste de datos
- Mejora la precisión del modelo.
- Se ocupa de datos de dimensiones superiores de manera eficiente

Aplicaciones

La técnica de boosting se utiliza en una gran cantidad de industrias, lo que proporciona información sobre el valor del mundo real y perspectivas interesantes. Los casos de uso clave incluyen:

- **Cuidado de la salud:** el embolsado se ha utilizado para formar predicciones de datos médicos. Por ejemplo, muestra que los métodos de conjunto se han utilizado para una variedad de

problemas de bioinformática, como la selección de genes o proteínas para identificar un rasgo específico de interés. Más específicamente, esto profundiza en su uso para predecir la aparición de diabetes en función de varios predictores de riesgo.

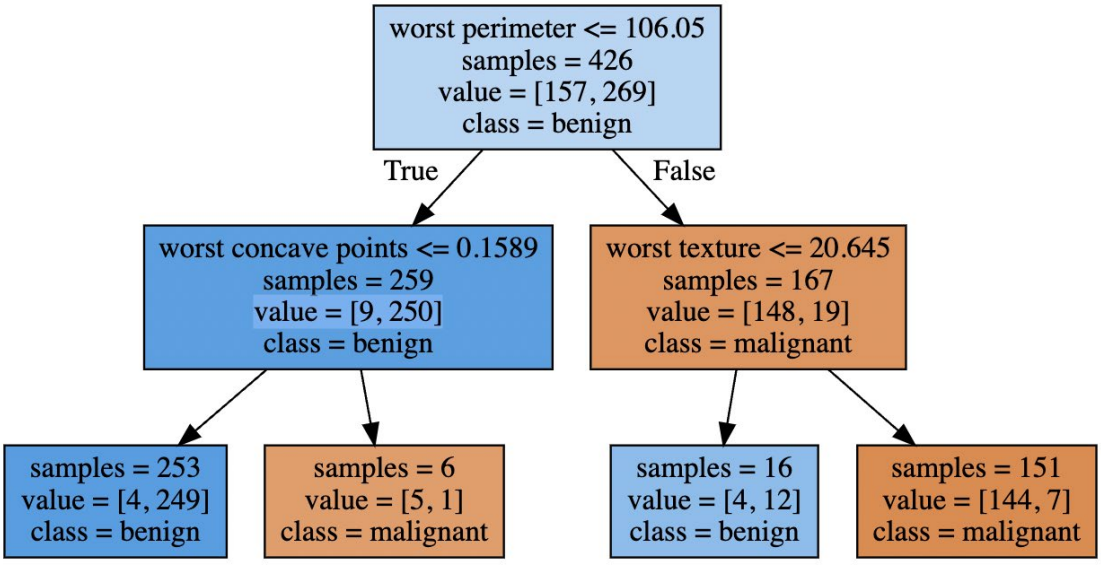
- **TI** : el embolsado también puede mejorar la precisión y la exactitud en los sistemas de TI, como los sistemas de detección de intrusos en la red. Mientras tanto, se analiza cómo el boosting puede mejorar la precisión de la detección de intrusos en la red y reducir las tasas de falsos positivos.
- **Medio ambiente** : Los métodos de conjunto, se han aplicado en el campo de la teledetección. Más específicamente, muestra cómo se ha utilizado para mapear los tipos de humedales dentro de un paisaje costero.
- **Finanzas** : el boosting también se ha aprovechado con modelos de aprendizaje profundo (deep learning) en la industria financiera, automatizando tareas críticas, incluida la detección de fraudes, evaluaciones de riesgo crediticio y problemas de fijación de precios de opciones.

Ejemplo en Python

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```

```
# Utilizando el dataset breast cancer

cancer = load_breast_cancer()




# dataset en formato tabular

pd.DataFrame(data=cancer.data,

            columns=cancer.feature_names).head()




# Separando los datos en sets de entrenamiento y
evaluación

X_train, X_test, y_train, y_test =
train_test_split(cancer.data,

                         cancer.target,
random_state=1)




# Armando un simple arbol de decisión

tree = DecisionTreeClassifier(max_depth=2,
random_state=0)

tree.fit(X_train, y_train)

print('Precisión modelo inicial train/test
{0:.3f}/{1:.3f}'
```

```
      .format(tree.score(X_train, y_train),
tree.score(X_test, y_test)))

# Precisión modelo inicial train/test  0.962/0.888


from sklearn import tree

# Dibujando el modelo

tree.export_graphviz(tree, out_file="tree.dot",
class_names=["malignant", "benign"], feature_names =
cancer.feature_names, impurity=False, filled=True)




with open("tree.dot") as f:

    dot_graph = f.read()

graphviz.Source(dot_graph)
```



```
# Utilizando AdaBoost para aumentar la precisión

ada = AdaBoostClassifier(base_estimator=tree,
n_estimators=500,

                         learning_rate=1.5,
random_state=1)

# Ajustando los datos

ada = ada.fit(X_train, y_train)

# Imprimir la precisión.

y_train_pred = ada.predict(X_train)

y_test_pred = ada.predict(X_test)

ada_train = accuracy_score(y_train, y_train_pred)

ada_test = accuracy_score(y_test, y_test_pred)

print('Precisión modelo con AdaBoost train/test
{0:.3f}/{1:.3f}'

      .format(ada_train, ada_test))

#Precisión modelo con AdaBoost train/test 1.000/0.965
```

Referencias

[1] Dónde aplicar boosting

https://www.ibm.com/cloud/learn/bagging

[2] Boosting

https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/

Material Complementario

[1] Qué es:

https://es.coursera.org/lecture/machine-learning-techniques/gradient-boosting-LsWkO

[2] Ejemplo

https://www.youtube.com/watch?v=3CC4N4z3GJc