# Ensamblaje en Machine Learning

# 1. Introducción al Ensamblaje

El ensamblaje en machine learning es una técnica que combina las predicciones de varios modelos (conocidos como modelos base o débiles) para generar una predicción más precisa y robusta. El objetivo principal es mejorar el rendimiento general, aprovechando la sabiduría colectiva. Un ejemplo intuitivo es cómo se toman decisiones grupales: la combinación de múltiples perspectivas a menudo produce mejores resultados que depender de un solo individuo.

En machine learning, esta técnica se utiliza principalmente para resolver problemas de clasificación y regresión. Los ensamblajes suelen ser más robustos frente a sobreajuste y ruido en los datos.

# 2. Características Principales del Ensamblaje

Las principalescaracterísticas del ensamblajeincluyen:

* **Mejora de la precisión**: al combinar múltiplesmodelos, se reducenerrores de predicción.
* **Robustez**: reduce el impacto de modelosdébiles o incorrectos.
* **Flexibilidad**: se pueden usar diversas técnicas como bagging, boosting y stacking.

Las desventajas que Podemos encontrarenestosprocesos:

* Mayor complejidad computacional
* Riesgo de sobreajuste si no se configura correctamente.

# 3. Cuándo Utilizar Ensamblaje

El ensamblaje es útilensituacionesdonde:

* El rendimiento de un modelo individual no es suficiente.
* Se dispone de datos con ruido o característicascomplejas.
* Se necesitamejorar la estabilidad y la precisión del modelo.

Por ejemplo, entareas de clasificación de imágenes o predicciónfinanciera.

# 4. Cómo Implementarlo

El ensamblaje se implementa utilizando bibliotecas como Scikit-learn. Aquí hay un ejemplopráctico de Random Forest. Enesteejemplo, el Random Forest combinamúltiplesárboles de decisión para mejorar la precisión.

from sklearn.ensemble import RandomForestClassifier  
from sklearn.datasets import load\_iris  
from sklearn.model\_selection import train\_test\_split  
  
# Cargardatos  
X, y = load\_iris(return\_X\_y=True)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Crear y entrenar el modelo  
model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)  
  
# Evaluar el modelo  
accuracy = model.score(X\_test, y\_test)  
print(f'Precisión: {accuracy:.2f}')

# ConsideracionesPrácticas

Aspectos a tenerencuenta al usar ensamblaje:

* Requiere mayor capacidadcomputacional.
* Puede ser complicadoajustar los hiperparámetros de todos los modelos base.
* Es importanteverificar la independencia de los modelos base para maximizarsuefectividad.

# TécnicasEspecíficas

## Clasificador por Votos - Voting Classifier

Un clasificador de votoscombina las predicciones de múltiplesmodelos base y elige la predicción final mediante:

* **Hard Voting**: Predicciónbasadaen la mayoría de votos (clasemásvotada).
* **Soft Voting**: Promedia las probabilidadespredichas y selecciona la clase con la probabilidadmásalta.

### Cuándo usar:

* Cuando se tienenvariosmodelos bien entrenados que son diversos (predicen de maneradiferente).
* Ideal para mejorar la estabilidad y la precisión general.

### Visualización:

Diagrama

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

### Implementaciónen Python:

from sklearn.ensemble import VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Cargar el dataset Iris

X, y = load\_iris(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Crear los modelos base

log\_clf = LogisticRegression(random\_state=42, max\_iter=1000)

svc\_clf = SVC(probability=True, random\_state=42)

rf\_clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Crear el Voting Classifier

voting\_clf = VotingClassifier(

estimators=[

('log\_reg', log\_clf),

('svc', svc\_clf),

('random\_forest', rf\_clf)

],

voting='hard' #Cambiar a 'soft' para promedio de probabilidades

)

# Entrenar el modelo de ensamble

voting\_clf.fit(X\_train, y\_train)

# Hacer predicciones

y\_pred = voting\_clf.predict(X\_test)

# Evaluarprecisión

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Precisión del clasificador por votos: {accuracy:.2f}")

## Bagging y Pasting

Ambos metodos entrenan multiples modelos base en subconjuntos aleatorios del conjunto de datos:

* **Bagging**: Muestreo con reemplazo (puede incluir duplicados en los subconjuntos).
* **Pasting**: Muestreo sin reemplazo (subconjuntos únicos).

### Cuándo usar:

* **Bagging**: Cuando el conjunto de datos es pequeño o se desea reducir la varianza.
* **Pasting**: Cuando el conjunto de datos es grande y se busca evitar duplicados.

### Visualización:

Diagrama

Descripción generada automáticamente

### Implementaciónen Python (Bagging y Pasting):

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Generardatossintéticos

X, y = make\_classification(n\_samples=100, n\_features=5, n\_classes=2, random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Pasting: muestreo sin reemplazo

pasting\_clf = BaggingClassifier(

base\_estimator=DecisionTreeClassifier(),

n\_estimators=10, # Número de predictors

# bootstrap=True, # Muestreo con reemplazo

bootstrap=False, # Muestreo sin reemplazo

random\_state=42

)

# Entrenamiento y predicción

pasting\_clf.fit(X\_train, y\_train)

y\_pred = pasting\_clf.predict(X\_test)

# Evaluar el rendimiento

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Precisión con Pasting: {accuracy:.2f}")

### Diferencias Clave

* **Bagging**:
  + bootstrap=True: Muestreo con reemplazo, permitiendo duplicados en los subconjuntos.
  + Es útil para reducir la varianzaen conjuntos de datospequeños o ruidosos.
* **Pasting**:
  + bootstrap=False: Muestreo sin reemplazo, asegurandosubconjuntosúnicos.
  + Ideal para conjuntos de datosmásgrandes y sin ruidosignificativo.

## Random Forest

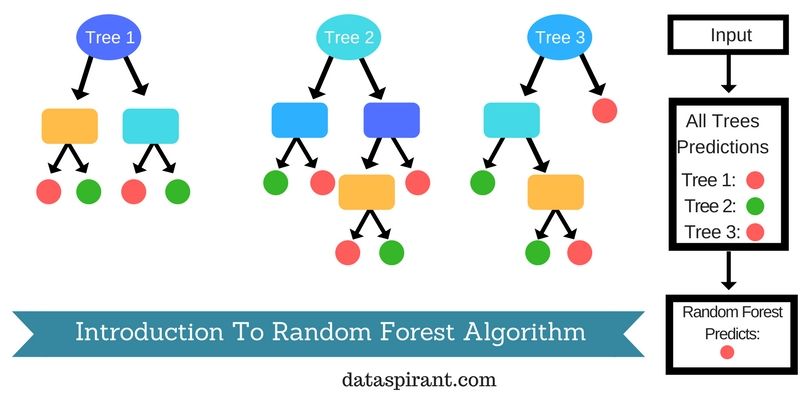
Random Forest es un ensamblaje de árboles de decision entrenados mediante bagging. Es eficiente y altamentepreciso, especialmente enclasificación y regresión.

* Característicasúnicas:
  + Se seleccionan subconjuntos aleatorios de características en cada división.
  + Reduce el sobreajuste y mejora la precisión.

### Cuándo usar:

* Enproblemas con alta dimensionalidad o conjuntos de datos ruidosos.
* Tanto para clasificación como para regresión.

### Visualización:



### Implementaciónen Python:

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make\_classification

import numpy as np

# Generar un conjunto de datos de ejemplo

X, y = make\_classification(n\_samples=100, n\_features=5, n\_classes=2, random\_state=42)

# Entrenar el modelo Random Forest

model = RandomForestClassifier(n\_estimators=3, random\_state=42) # 3 árboles

model.fit(X, y)

# Crear un ejemplo de entrada

input\_data = np.array([[0.5, -1.2, 0.3, 0.8, -0.5]]) # Cambia estosvalorescomoprefieras

# Prediccionesindividuales de cada árbol

tree\_predictions = [tree.predict(input\_data) for tree in model.estimators\_]

print(f"Predicciones de los árbolesindividuales: {tree\_predictions}")

# Predicción final del Random Forest

final\_prediction = model.predict(input\_data)

print(f"Predicción final del Random Forest: {final\_prediction[0]}")

## Boosting

Boosting combina modelos secuenciales, donde cada modelo intent corregir los errores del anterior. Ejemplos:

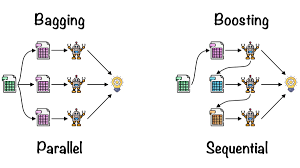
* **AdaBoost**: Asigna pesos a las instancias mal clasificadas.
* **Gradient Boosting**: Optimiza los errors residuales.

### Cuándo usar:

* Cuando se necesitamejorar el rendimiento con modelos base débiles.
* Enproblemasdonde los errorestienen un patrón que puedecorregirse.

### Visualización:

***Comparación Baggin / Boosting***



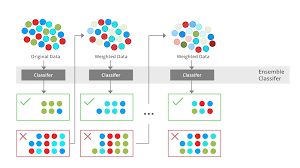
### Comparación

| **Aspecto** | **Bagging (Paralelo)** | **Boosting (Secuencial)** |
| --- | --- | --- |
| **Ejecución** | Paralelo: los modelos se entrenansimultáneamente. | Secuencial: cadamodelodepende del anterior. |
| **Objetivo** | Reducir la varianza. | Reducir el sesgo. |
| **Muestreo de Datos** | Muestreo con reemplazo (bootstrap). | Usocompleto del conjunto, ajustando pesos. |
| **Combinación** | Votación (clasificación) o promedio (regresión). | Ponderaciónbasadaen el rendimiento. |
| **Ejemplo** | Random Forest. | AdaBoost, Gradient Boosting. |

### ¿Cuál elegir?

* **Bagging**: Útilsitumodelo base tiende a sobreajustarse (reduce varianza).
* **Boosting**: Mejor cuandotumodelo base tiene un sesgo alto y quieresaumentarsuprecisión.

#### Boosting



Esta imagen ilustra el proceso de **Boosting**, un método de ensamblaje secuencial que busca mejorar el rendimiento de un modelo combinando múltiples clasificadores débiles en un clasificador fuerte. A continuación, te explico cada parte:

### ****1. Conjunto de Datos Original****

* Se parte de un conjunto de datos original, dondetodas las instanciastienen el mismo peso inicial.
* El primer clasificador se entrena con este conjunto de datos.

### ****2. Pesado de los Datos****

* Trasentrenar el primer clasificador:
  + Las instancias que se clasificaroncorrectamentereciben un peso menor.
  + Las instancias mal clasificadasreciben un peso mayor.
* Este ajuste de pesos asegura que el siguienteclasificador se enfoqueen los datosdifíciles.

### ****3. ClasificadoresSecuenciales****

* Cadaclasificadoren la secuencia se entrena con los datosreponderados, es decir, aquellosdatos que son másdifíciles de clasificartienenmásrelevanciaen las iteracionessiguientes.
* Este proceso se repite para cadaclasificador.

### ****4. Combinación Final (Ensemble Classifier)****

* Una vez que todos los clasificadoreshansidoentrenados, sus predicciones se combinan.
* La combinaciónpuederealizarsemediante un promedioponderado o una votaciónponderada.
* El resultado final es una predicciónmásprecisa, ya que se aprovechan los puntos fuertes de cadaclasificador.

### ****Boosting enAcción****

Un ejemplo práctico es **AdaBoost**:

1. Entrena un clasificadordébil (como un árbol de decisión con profundidad 1).
2. Ajusta los pesos de las instanciasenfunción de los errores.
3. Repite el proceso para un númerofijo de clasificadores.
4. Combina los resultados de los clasificadoresutilizando sus pesos de precisión.

### Implementaciónen Python (AdaBoost y Gradient Boosting):

*1. Bagging (Parallel)*

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Generardatossintéticos

X, y = make\_classification(n\_samples=100, n\_features=5, n\_classes=2, random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Bagging (Ejecución Paralela)

bagging\_clf = BaggingClassifier(

base\_estimator=DecisionTreeClassifier(),

n\_estimators=10, # Número de predictores

bootstrap=True, # Muestreo con reemplazo

n\_jobs=-1, # Paralelización

random\_state=42

)

# Entrenamiento y predicción

bagging\_clf.fit(X\_train, y\_train)

y\_pred = bagging\_clf.predict(X\_test)

# Evaluar el rendimiento

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Precisión con Bagging (Parallel): {accuracy:.2f}")

*Explicación:*

* BaggingClassifiercreamúltiplesárboles de decisión.
* n\_jobs=-1permiteparalelización, utilizandotodos los núcleosdisponibles para entrenar los predictoressimultáneamente.

*2. Boosting (Sequential)*

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Generardatossintéticos

X, y = make\_classification(n\_samples=100, n\_features=5, n\_classes=2, random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Boosting (Ejecución Secuencial)

boosting\_clf = AdaBoostClassifier(

base\_estimator=DecisionTreeClassifier(max\_depth=1), #Árbolesdébiles

n\_estimators=10, # Número de predictoressecuenciales

learning\_rate=1.0, # Peso de cadamodelo

random\_state=42

)

# Entrenamiento y predicción

boosting\_clf.fit(X\_train, y\_train)

y\_pred = boosting\_clf.predict(X\_test)

# Evaluar el rendimiento

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Precisión con Boosting (Sequential): {accuracy:.2f}")

*Explicación:*

* AdaBoostClassifierentrenasecuencialmenteárbolesdébiles (profundidad 1).
* Cadamodelo posterior se ajusta para corregir los errores de los modelosanteriores.

### Diferencias Clave

| **Método** | **Característica** | **Ejecución** | **Clase Usada** |
| --- | --- | --- | --- |
| **Bagging** | Modelosentrenadosenparalelo | Paralelo | BaggingClassifier |
| **Boosting** | Modelosajustadossecuencialmente | Secuencial | AdaBoostClassifier |

## Stacking

Stacking utiliza un modelo meta-aprendiz (blender) para combinarpredicciones de múltiplesmodelos base.

* Es una técnicapoderosa para mejorar el rendimientoentareascomplejas.
* Mejora el rendimiento al aprovechar las fortalezas de cadamodelo base.

### Cuándo usar:

* Cuando los modelos base tienenpatronescomplementarios.
* Ideal para problemascomplejosdonde una combinación simple (e.g., voting) no es suficiente.

### Visualización:

Diagrama

Descripción generada automáticamente

### Implementaciónen Python:

fromsklearn.ensembleimportStackingClassifier

# Modelos base

base\_estimators = [

('lr', LogisticRegression()),

('svc', SVC(probability=True)),

('dt', DecisionTreeClassifier())

]

# Meta-modelo

stacking\_clf = StackingClassifier(

estimators=base\_estimators,

final\_estimator=LogisticRegression(),

passthrough=True # Incluye características originales en el meta-modelo

)

stacking\_clf.fit(X\_train, y\_train)

accuracy\_stacking = stacking\_clf.score(X\_test, y\_test)

print(f"Precisión con Stacking: {accuracy\_stacking:.2f}")

## Resumen Comparativo

| **Técnica** | **Características Clave** | **Uso Principal** | **Clase de Scikit-learn** | **Temas Importantes a Considerar** |
| --- | --- | --- | --- | --- |
| **Voting** | Combinación de predicciones finales mediante hard/softvoting | Mejorar estabilidad y precisión | VotingClassifier | - Los modelos base deben ser diversos para evitar redundancia.  - "Softvoting" requiere modelos con predict\_proba. |
| **Bagging/Pasting** | Subconjuntos aleatorios de datos | Reducir varianza (Bagging) | BaggingClassifier, BaggingRegressor | - Bagging es mejor para datos pequeños y ruidosos.  - Pasting funciona bien con conjuntos grandes.  - La independencia de los modelos base mejora los resultados. |
| **Random Forest** | Subconjuntos de datos y características aleatorios | Clasificación y regresión complejas | RandomForestClassifier, RandomForestRegressor | - Escalable con grandes cantidades de datos.  - Sensible a hiperparámetros como max\_depth y n\_estimators.  - Reduce sobreajuste comparado con árboles de decisión simples. |
| **Boosting** | Corrección secuencial de errores | Incrementar precisión | AdaBoostClassifier, AdaBoostRegressor, GradientBoostingClassifier, GradientBoostingRegressor | - AdaBoost es más sensible al ruido en los datos.  - GradientBoosting puede ajustarse demasiado; regularización como learning\_rate y n\_estimators es clave.  - Costoso computacionalmente para conjuntos de datos grandes. |
| **Stacking** | Meta-aprendiz que combina modelos base | Mejorar rendimiento en problemas complejos | StackingClassifier, StackingRegressor | - Seleccionar un buen meta-modelo es crítico.  - La diversidad entre los modelos base mejora la precisión.  - passthrough=True puede incluir características originales para más contexto. |

# Hiperparámetros Clave

Los hiperparámetros entécnicas de ensamblaje son esenciales para controlar el rendimiento de los modelos, ya que afectansuprecisión, velocidad y robustez. A continuación, se detallan los másrelevantes y cómoconfigurarlos:

### Hiperparámetros

* **Número de Estimadores (n\_estimators)**
* **Descripción**: Determina la cantidad de modelos base que se entrenaránen el ensamble (por ejemplo, árbolesen Random Forest o iteracionesen AdaBoost).
* **Impacto**:
  + **Mayor número**:
    - Mejora la precisión al reducir la varianza o sesgo, peroaumenta el tiempo de entrenamiento.
    - Puedeprovocarsobreajusteenalgunoscasos, especialmente con boosting.
  + **Menor número**:
    - Reduce el tiempo de entrenamiento, peropuedegenerarmodelossubóptimos.
* **Valoresrecomendados**:
  + Random Forest: Entre 100 y 500.
  + Boosting (AdaBoost/Gradient Boosting): Comienza con 50-100.
* **Muestreo (bootstrap y bootstrap\_features)**
* **Bootstrap (bootstrap)**:
  + Indica si se realizamuestreo con reemplazo para los subconjuntos de entrenamiento.
    - **True**: Usadoen Bagging y Random Forest; permiteduplicados, mejor para datospequeños.
    - **False**: Usadoen Pasting, mejor para datosgrandes y ricos.
* **Muestreo de Características (bootstrap\_features)**:
  + Define si las características (columnas) debenseleccionarsealeatoriamente con reemplazo.
  + Muy útilen Random Forest para reducir la correlación entre árboles.
* **Profundidad Máxima de los Modelos Base (max\_depth)**
* **Descripción**: Controla la complejidad de cadamodelo base (por ejemplo, árboles de decisión).
* **Impacto**:
  + **Mayor profundidad**:
    - Capturarelacionesmáscomplejas, peroaumenta el riesgo de sobreajuste.
  + **Menor profundidad**:
    - Generalizamejor, peropuedesufrir de infraajuste.
* **Valoresrecomendados**:
  + Random Forest: Prueba con valores entre 5 y 20.
  + Boosting: Mantén los modelosdébiles, comomax\_depth=3.
* **Fracción de Datos Usados en el Muestreo (max\_samples)**
* **Descripción**: Porcentaje de datosseleccionados para entrenarcadamodelo base.
* **Impacto**:
  + **Mayor fracción**:
    - Mejora la precisión, peroaumenta la correlación entre modelos base.
  + **Menor fracción**:
    - Reduce la correlación, ideal para datosmuygrandes.
* **Valoresrecomendados**:
  + Comienza con 0.5-0.8 y ajustasegún los resultados.
* **Tasa de Aprendizaje (learning\_rate) (para Boosting)**
* **Descripción**: Determinacuánto peso se da a cadamodelo base en el boosting.
* **Impacto**:
  + **Alta tasa de aprendizaje**:
    - Converge rápidamente, peropuedeperderdetalles.
  + **Baja tasa de aprendizaje**:
    - Convergenciamáslenta, peromejora la precisión al final.
* **Valoresrecomendados**:
  + AdaBoost/Gradient Boosting: 0.01 a 0.1.
* **Selección de Características (max\_features)**
* **Descripción**: Númeromáximo de característicasconsideradas al entrenarcadamodelo base.
* **Impacto**:
  + **Menor número**:
    - Reduce la correlación entre modelos base (mejoraen Random Forest).
  + **Mayor número**:
    - Capturamásinformación, peropuedeaumentar la correlación.
* **Valoresrecomendados**:
  + Random Forest:
    - Clasificación: √(número de características).
    - Regresión: Entre 1/3 y 2/3 del total de características.
* **Penalización para Boosting (subsample)**
* **Descripción**: Proporción de datosusada para entrenarcadaiteración de Boosting.
* **Impacto**:
  + **Menor fracción**:
    - Reduce el sobreajuste al aumentar la diversidad entre iteraciones.
  + **Mayor fracción**:
    - Mejora la estabilidad, peropuedeaumentar el riesgo de sobreajuste.
* **Valoresrecomendados**:
  + Comienza con 0.5-0.8.
* **Peso de las Clases (class\_weight)**
* **Descripción**: Controlacómo se penalizan las clasesdesbalanceadas.
* **Impacto**:
  + Mejora la precisión para clasesminoritariasen conjuntos desbalanceados.
* **Opciones**:
  + balanced: Calculaautomáticamente pesos basadosen la proporción de clases.
  + Diccionario: Especifica pesos manualmente.
* **Número de Hilos Paralelos (n\_jobs)**
* **Descripción**: Número de núcleos CPU utilizados para entrenar los modelos base.
* **Impacto**:
  + **-1**: Utilizatodos los núcleosdisponibles (recomendado para grandesdatos).
  + Acelera el entrenamiento de Bagging y Random Forest.

### CómoAjustar Hiperparámetros

* **Validación Cruzada**:
  + UsaGridSearchCV o RandomizedSearchCV para encontrar los mejoresvalores.
* **Empieza con Valores por Defecto**:
  + Ajustamanualmente solo los hiperparámetros mássensibles (comon\_estimators y max\_depth).
* **Evita el Sobreajuste**:
  + Usavalidacióncruzada y pruebas con datos no vistos.