# ¿Qué es Random Forest?

1

## Algoritmo de Aprendizaje Automático

Random Forest es un algoritmo de aprendizaje supervisado que se basa en la construcción de múltiples árboles de decisión.

2

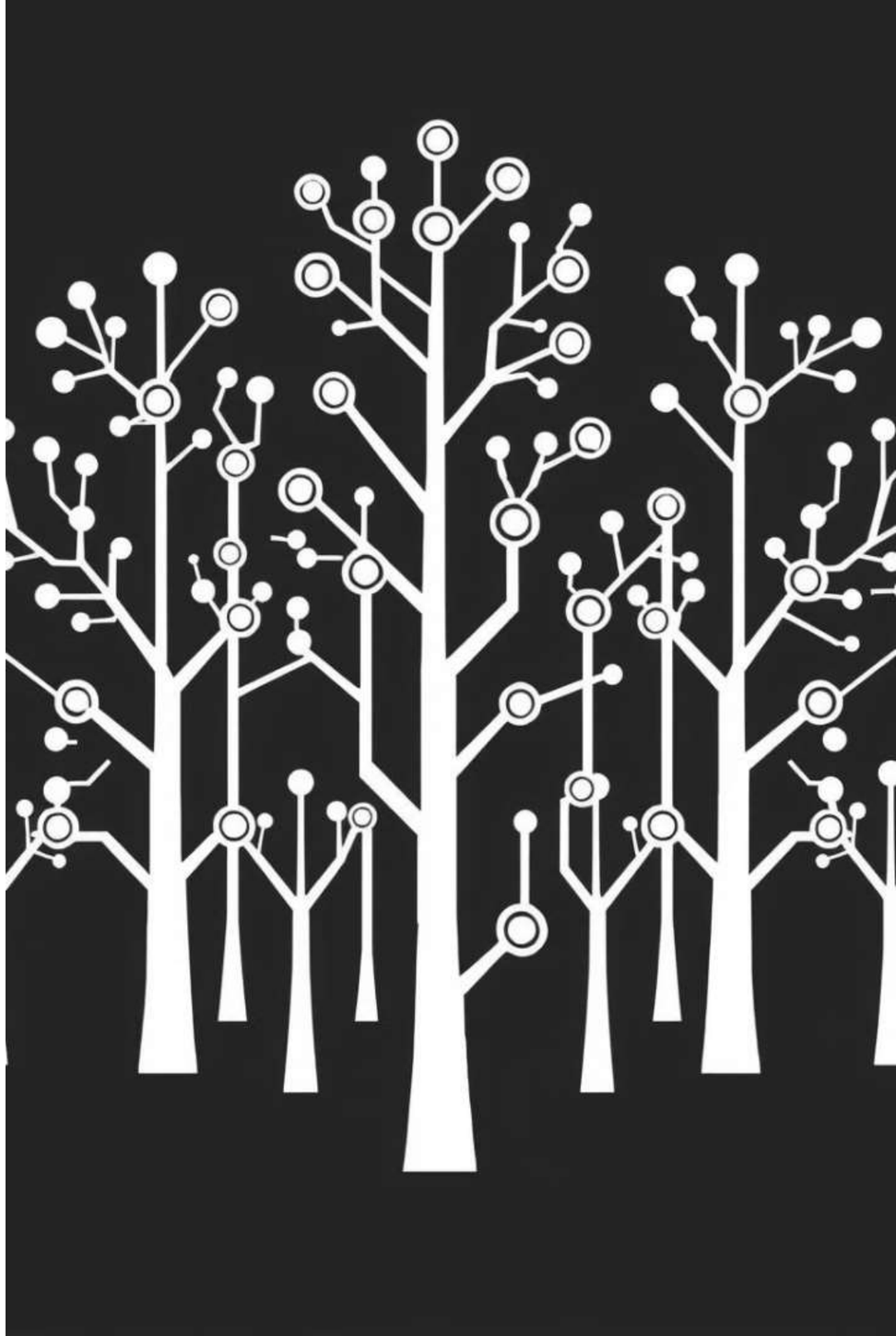
## Clasificación y Regresión

Puede utilizarse tanto para problemas de clasificación como de regresión, dependiendo de la naturaleza de la variable objetivo.

3

## Ensemble Learning

Combina las predicciones de múltiples árboles de decisión para obtener resultados más robustos y precisos.



# Cómo funciona el método Random Forest

1

## Construcción de Árboles

Random Forest genera múltiples árboles de decisión a partir de submuestras aleatorias del conjunto de datos original.

2

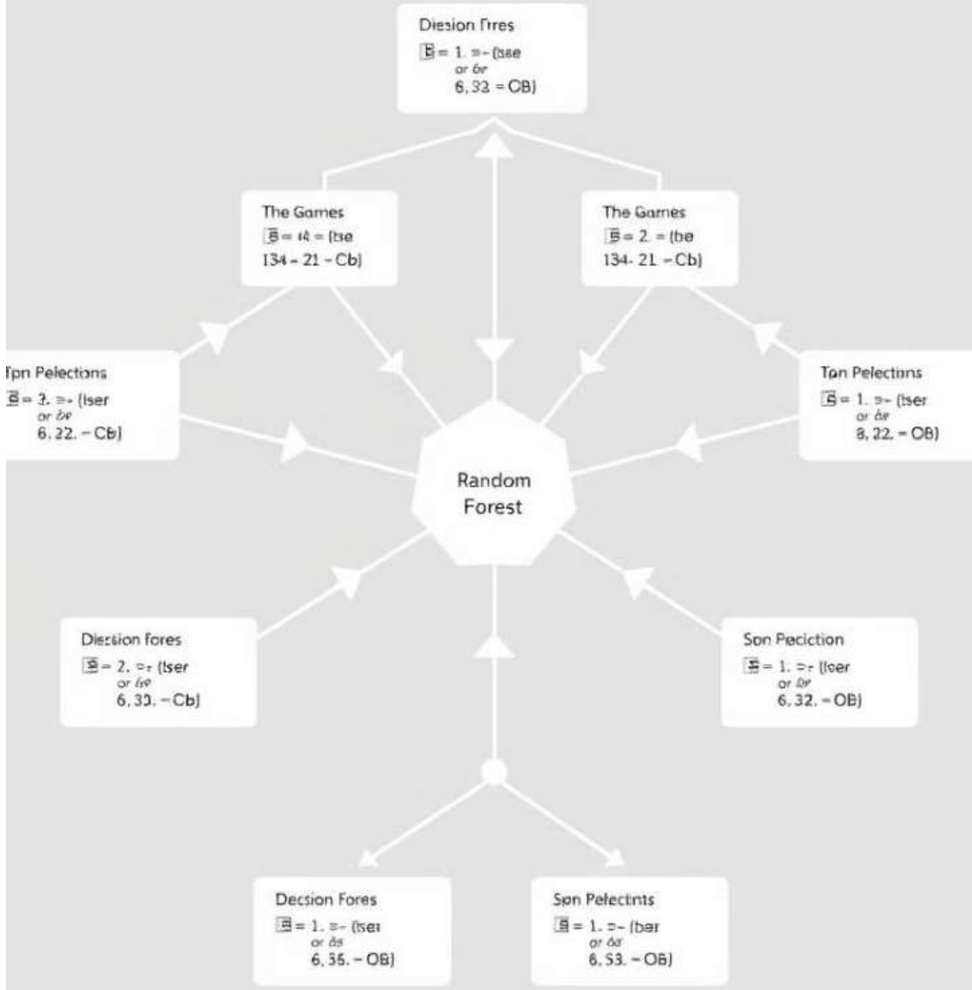
## Entrenamiento

Cada árbol se entrena de manera independiente utilizando un subconjunto diferente de las variables predictoras.

3

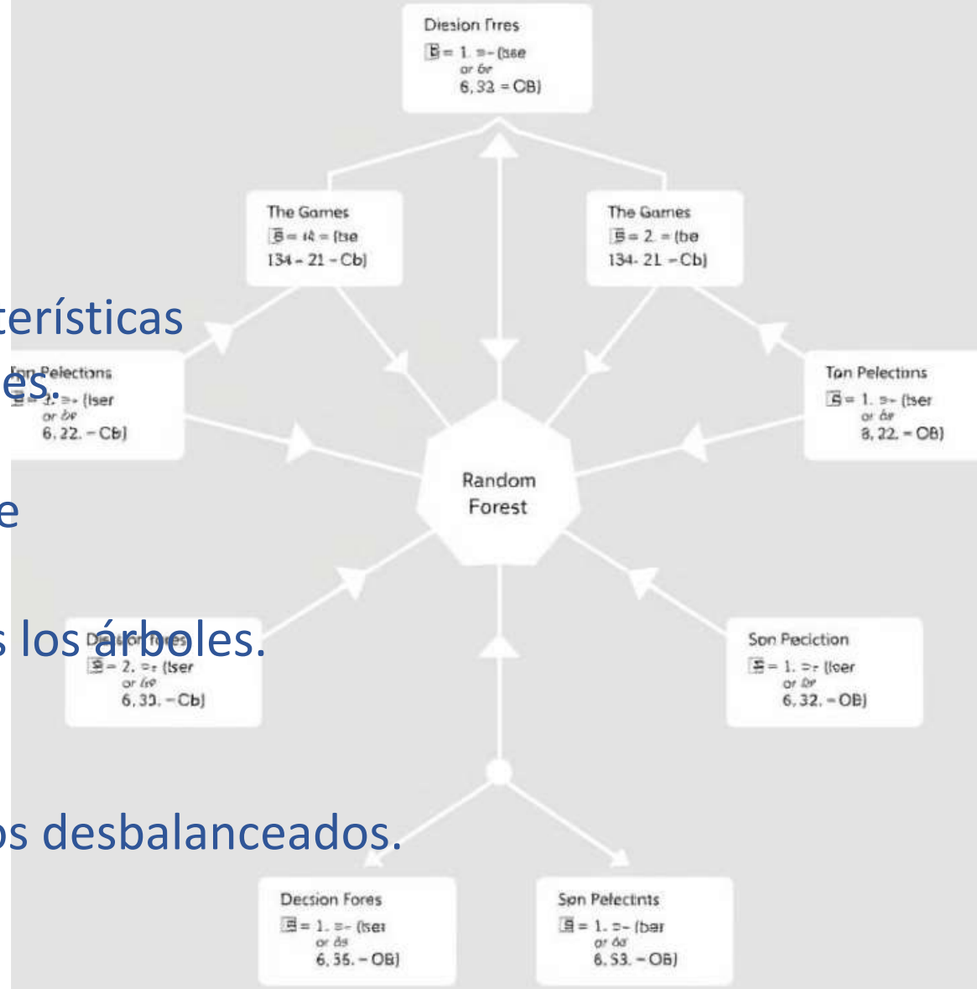
## Votación/Promedio

Las predicciones finales se obtienen mediante la votación mayoritaria (clasificación) o el promedio (regresión) de las predicciones de todos los árboles.



# Cómo funciona el método Random Forest

1. **Construcción del bosque**:
   - **Bootstrap Aggregation (Bagging)**:
     Cada árbol de decisión se entrena en un subconjunto aleatorio de los datos de entrenamiento. Este proceso se denomina "muestreo con reemplazo".
   - **Selección aleatoria de características**:
     En cada nodo, se selecciona un subconjunto aleatorio de características para dividir los datos. Esto introduce diversidad entre los árboles.
2. **Predicción**:
   - Para clasificación: Cada árbol emite un voto por una clase, y la clase final se decide por mayoría (votación).
   - Para regresión: Se calcula el promedio de las predicciones de todos los árboles.
3. **Ventajas**:
   - Reduce el riesgo de sobreajuste gracias a la aleatoriedad.
   - Funciona bien con características altamente correlacionadas y datos desbalanceados.
   - Capaz de manejar datos faltantes o incompletos.
4. **Limitaciones**:
   - Puede ser más lento y consumir más memoria que los árboles de decisión simples.
   - Menos interpretable que un único árbol de decisión.



# Parámetros de Random Forest

**1. n_estimators (int, default=100)**
Número de árboles en el bosque.
Más árboles suelen mejorar el rendimiento, pero también aumentan el tiempo de entrenamiento y predicción.
**2. criterion (str, default='gini' para clasificación, 'squared_error' para regresión)**
Función para medir la calidad de la división:

- Clasificación:
  - 'gini': Impureza de Gini.
  - 'entropy': Ganancia de información.
- Regresión:
  - 'squared_error': Error cuadrático medio (MSE).
  - 'absolute_error': Error absoluto medio (MAE).

**3. max_depth (int, default=None)**
Profundidad máxima de los árboles.
None permite que los árboles crezcan hasta que todas las hojas sean puras o contengan menos muestras que min_samples_split.

**4. min_samples_split (int or float, default=2)**
Mínimo número de muestras requeridas para dividir un nodo.
Si es un flotante, representa un porcentaje del total (e.g., 0.05 significa el 5%).
**5. min_samples_leaf (int or float, default=1)**
Mínimo número de muestras requeridas en una hoja.
Ayuda a evitar que el modelo se sobreajuste en datos ruidosos.
**6. min_weight_fraction_leaf (float, default=0.0)**
Fracción mínima del peso total de las muestras requerida en una hoja.

# Parámetros de Random Forest

**7. max_features (int, float, str, or None, default='sqrt')**
Número máximo de características consideradas para dividir un nodo:
- int: Número exacto de características.
- float: Porcentaje de características.
- 'sqrt': Raíz cuadrada del total de características (por defecto en clasificación).
- 'log2': Logaritmo base 2 del total de características.
- None: Usa todas las características.

**8. max_leaf_nodes (int, default=None)**
Número máximo de nodos hoja. Limitarlo puede hacer el modelo más simple y rápido.

**9. min_impurity_decrease (float, default=0.0)**
Umbral para dividir un nodo. Solo se divide si la reducción en la impureza es mayor que este valor.

**10. bootstrap (bool, default=True)**
Si se utiliza muestreo con reemplazo para construir cada árbol.
True: Se realiza muestreo con reemplazo.
False: Usa todo el conjunto de datos.

**11. oob_score (bool, default=False)**
Habilita la evaluación con muestras fuera de la bolsa (*out-of-bag*), útil para validación cruzada interna.

**12. n_jobs (int, default=None)**
Número de núcleos de CPU usados para entrenamiento:
- -1: Usa todos los núcleos disponibles.
- Un entero positivo indica el número de núcleos.

**13. random_state (int, default=None)**
Fija la semilla para garantizar reproducibilidad de resultados.

**14. verbose (int, default=0)**
Controla la salida de información del proceso:
- 0: Sin salida.
- 1: Información básica.
- >1: Información más detallada.

**15. warm_start (bool, default=False)**
Si se reutilizan soluciones previas al agregar árboles.
Útil para ajustar modelos progresivamente.

# Ejemplo en python

```
# Importar librerías necesarias
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
import pandas as pd

# Cargar el conjunto de datos Iris
iris = load_iris()
X = iris.data  # Características
y = iris.target  # Etiquetas

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Crear el modelo Random Forest
rf_model = RandomForestClassifier(
    n_estimators=100,    # Número de árboles en el bosque
    max_depth=5,         # Profundidad máxima de los árboles
    random_state=42,     # Asegurar reproducibilidad
    n_jobs=-1            # Usar todos los núcleos disponibles
)

# Entrenar el modelo
rf_model.fit(X_train, y_train)

# Predecir etiquetas para el conjunto de prueba
y_pred = rf_model.predict(X_test)

# Evaluar el modelo
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nReporte de clasificación:\n", classification_report(y_test, y_pred))
```

```
# Importancia de las características
feature_importances = rf_model.feature_importances_
feature_names = iris.feature_names

# Crear un gráfico de importancia de características
plt.barh(feature_names, feature_importances, color='skyblue')
plt.xlabel('Importancia') plt.ylabel('Características')
plt.title('Importancia de las características en Random Forest')
plt.show()
```

# Ejemplo

## De configuración de parámetros

```
from sklearn.ensemble import RandomForestClassifier

# Crear el modelo con parámetros ajustados
rf_model = RandomForestClassifier(
    n_estimators=200,          # Usar 200 árboles
    max_depth=10,              # Limitar la profundidad a 10 niveles
    max_features='sqrt',       # Seleccionar sqrt(n_features) en cada nodo
    min_samples_split=5,       # Dividir nodos con al menos 5 muestras
    bootstrap=True,            # Usar muestreo con reemplazo
    oob_score=True,            # Evaluar el rendimiento con muestras OOB
    random_state=42,           # Reproducibilidad
    n_jobs=-1                  # Usar todos los núcleos disponibles
)

# Entrenar el modelo
rf_model.fit(X_train, y_train)
```

## De optimización de parámetros

```
# Usa GridSearchCV o RandomizedSearchCV para encontrar los mejores parámetros

from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'max_features': ['sqrt', 'log2']
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Mejores parámetros:", grid_search.best_params_)
```

**Nota**

Los mismos parámetros se aplican a RandomForestRegressor, excepto por las métricas de error que son específicas de la regresión (criterion).
Estos parámetros te permiten ajustar el modelo según las necesidades de tu problema.

# Ventajas del algoritmo Random Forest

## Alta Precisión

El uso de múltiples árboles de decisión mejora significativamente la precisión de las predicciones. y evitar el sobreajuste (*overfitting*).

## Robustez ante el Ruido

Random Forest es poco sensible a la presencia de valores atípicos o datos ruidosos en los conjuntos de entrenamiento.

## Escalabilidad

Puede manejar eficientemente grandes volúmenes de datos y gran número de variables predictoras.

# Caso práctico: Clasificación de datos con Random Forest

### Descripción del Problema

Clasificar clientes según su probabilidad de suscribirse a un nuevo producto bancario.

### Datos de Entrenamiento

Variables como edad, ingresos, historial de compras y navegación web del cliente.

### Objetivo

Predecir si un cliente tiene alta o baja probabilidad de suscribirse al producto.

### Beneficios

Focalizar los esfuerzos de marketing en los clientes con mayor probabilidad de compra.

# Preparación de los datos

1

### Limpieza

Identificar y tratar valores faltantes, atípicos y ruido en los datos.

2

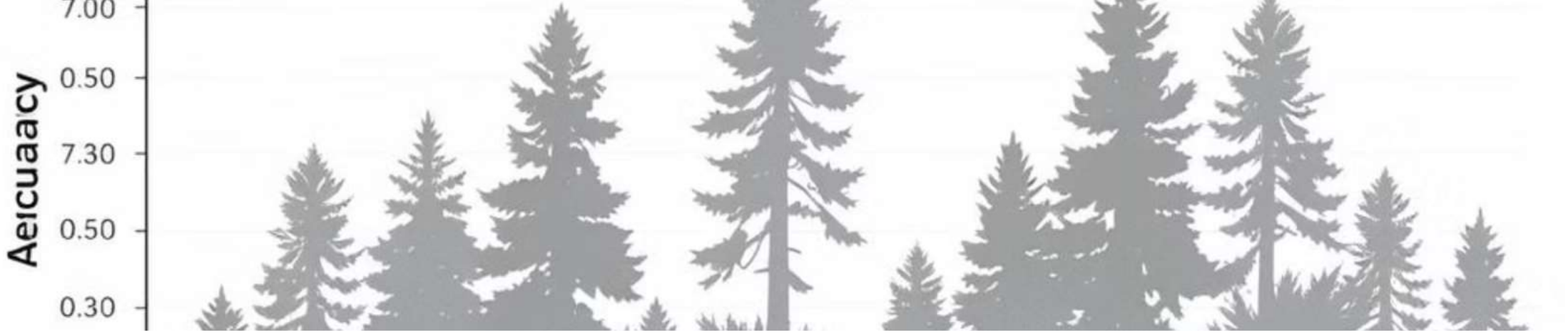
### Transformación

Convertir variables categóricas en numéricas y escalar las características.

3

### Selección de Variables

Identificar las características más relevantes para el modelo de clasificación.




# Análisis de resultados y conclusiones

| | |
|---|---|
| Precisión | 82% |
| Recall | 78% |
| F1-Score | 80% |
| AUC-ROC | 0.87 |

Los resultados muestran que el modelo de Random Forest ha logrado una alta precisión y capacidad predictiva en la clasificación de clientes con probabilidad de suscribirse al producto. Esto demuestra la eficacia del algoritmo para este tipo de problemas de aprendizaje supervisado.