MÓDULO II

Aprendizaje Supervisado

Curso:

Machine Learning Supervisado

🌲 Sesión 02: La Revolución No-Lineal

Árboles, Ensembles y el Camino hacia el Boosting

Del árbol solitario al bosque invencible

📋 Agenda de Hoy

Bloque	Tema	Duración
1	🌀 El Problema de la Linealidad	15 min
2	🌳 Árboles de Decisión (CART)	25 min
3	🌲 Random Forest (Bagging)	20 min
4	🚀 Gradient Boosting (XGBoost, LightGBM)	25 min
☕	**Break**	15 min
5	📐 SVM y KNN: Los Clásicos Geométricos	20 min
6	🥊 Arena de Combate: RF vs LightGBM vs XGBoost	40 min

🎯 Objetivos de Aprendizaje

Al finalizar esta sesión podrás:

1. **Romper la Linealidad:** Entender cuándo $y = mx + b$ no es suficiente
2. **Dominar Árboles:** Construir, podar e interpretar árboles de decisión
3. **Ensembles:** Diferenciar Bagging (RF) vs Boosting (XGBoost/LightGBM)
4. **Comparar Algoritmos:** Evaluar trade-offs de precisión, velocidad e interpretabilidad
5. **Aplicar en Datos Reales:** Implementar pipelines completos de clasificación

BLOQUE 1

El Problema de la Linealidad

¿Por qué la Regresión Logística no es suficiente?

🌀 El Límite de los Modelos Lineales

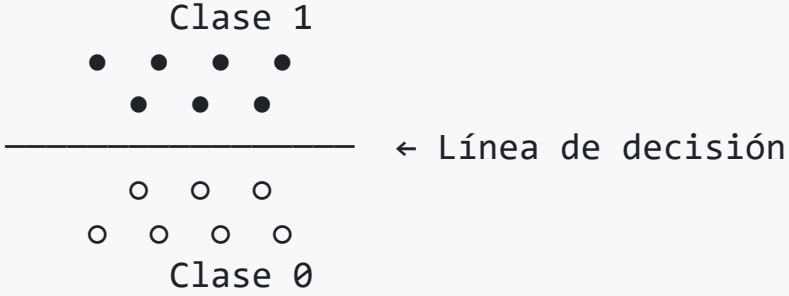
La Regresión Logística

- Dibuja una **línea recta** (hiperplano)
- Funciona cuando las clases son **linealmente separables**
- Falla con patrones complejos

¿Cuándo NO funciona?

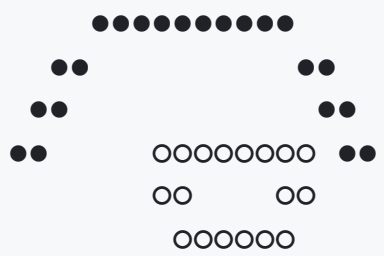
- Datos en forma de "lunas" 🌙
- Patrones circulares o espirales
- Interacciones complejas entre variables

Frontera de Decisión Lineal



> **Problema:** ¿Qué pasa si las clases están "entrelazadas"?

🌙 El Dataset "Moons": Un Clásico



- **Regresión Logística:** Accuracy ~86% (no puede separar las curvas)
- **Necesitamos:** Algoritmos que dibujen **fronteras curvas**

> *"Cuando tus datos tienen forma de luna, necesitas un modelo que entienda la luna."*

🌳 BLOQUE 2

Árboles de Decisión (CART)

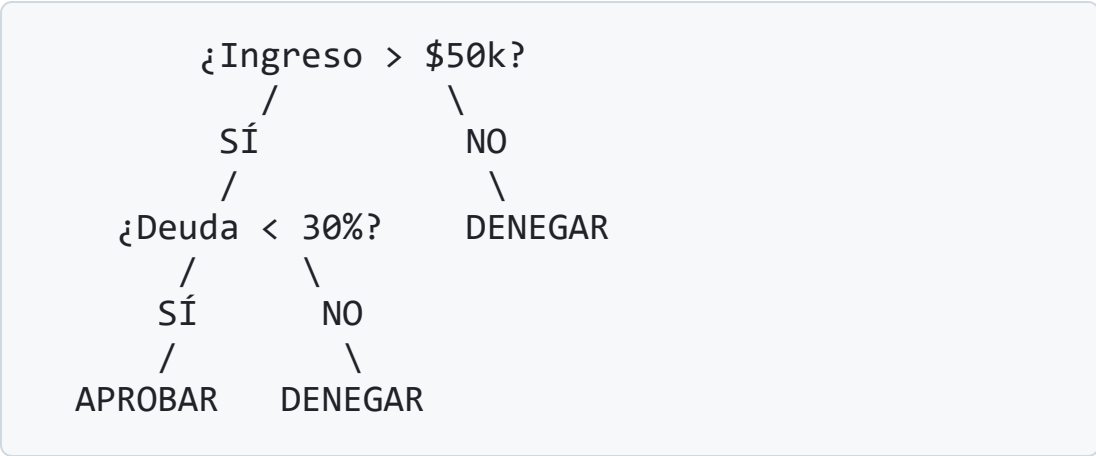
"Divide y Vencerás"

🌳 ¿Cómo Funciona un Árbol de Decisión?

El Algoritmo CART

1. **Pregunta:** "¿Feature X > valor?"
2. **Divide:** Separa los datos en dos grupos
3. **Repite:** Hasta que las hojas sean "puras"

Ejemplo: ¿Aprobar Crédito?



Ventajas

- ✅ **Muy interpretables** (reglas If-Then)
- ✅ No requieren escalado de datos
- ✅ Capturan no-linealidades
- ✅ Manejan categóricas naturalmente

Desventajas

- ❌ **Overfitting severo** sin control
- ❌ Fronteras "escalonadas"
- ❌ Inestables (pequeños cambios = árbol diferente)

⚠️ El Peligro del Overfitting

Árbol SIN Restricciones

```
tree = DecisionTreeClassifier(
    max_depth=None  # ❌ Sin límite
)
```

- **Accuracy en Train:** 100% 🎉
- **Accuracy en Test:** 75% 😱
- **Problema:** Memorizó el ruido

Árbol CON Poda (Pruning)

```
tree = DecisionTreeClassifier(
    max_depth=4,            # ✅ Límite
    min_samples_leaf=5      # ✅ Mínimo
)
```

- **Accuracy en Train:** 92%
- **Accuracy en Test:** 90% ✅
- **Resultado:** Generaliza mejor

> 💡 **Regla de Oro:** Si tu árbol tiene más hojas que $\sqrt{n}$, probablemente está memorizando.

🔑 Hiperparámetros Clave de Árboles

Parámetro	Descripción	Efecto
`max_depth`	Profundidad máxima	↓ profundidad = ↓ overfitting
`min_samples_split`	Mínimo para dividir	↑ valor = más conservador
`min_samples_leaf`	Mínimo en hojas	Evita hojas con 1-2 ejemplos
`max_features`	Features por split	Añade aleatoriedad
`criterion`	Métrica de pureza	`gini` (default) o `entropy`

Criterio de División: Gini vs Entropy

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 \qquad \text{Entropy} = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

🌲 BLOQUE 3

Random Forest (Bagging)

"Un experto se equivoca, mil promediados no"

🌲 Random Forest: La Democracia de los Árboles

¿Qué es Bagging?

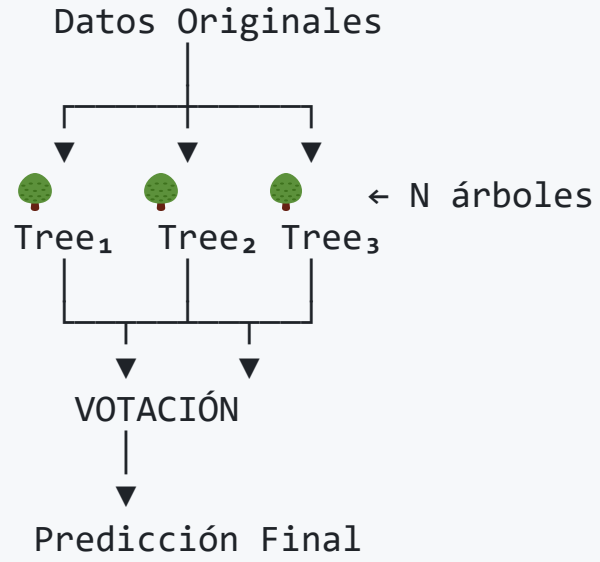
Bootstrap **Agg**regat**ing**:

1. Crear múltiples muestras bootstrap
2. Entrenar un árbol en cada muestra
3. Promediar las predicciones

Random Forest = Bagging + Aleatorización

- Cada árbol ve **subconjunto de datos**
- Cada split considera **subconjunto de features**
- **Resultado:** Árboles decorrelacionados

Diagrama Conceptual



🌲 Random Forest en Código

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=200,        # 200 árboles
    max_depth=10,            # Limitar profundidad
    min_samples_leaf=5,      # Mínimo por hoja
    max_features='sqrt',     # √n features por split
    class_weight='balanced', # Balanceo de clases
    n_jobs=-1,               # ¡Paralelizar!
    random_state=42
)

rf.fit(X_train, y_train)
```

> 💡 **Pro-Tip:** Con `n_jobs=-1` aprovechas **todos los cores** de tu CPU. Random Forest es embarazosamente paralelo.

🎯 ¿Por Qué Funciona Random Forest?

El Poder de la Diversidad

Un Solo Árbol	Random Forest
Alta varianza	Baja varianza
Inestable	Robusto
Puede memorizar	Promedia el ruido
Frontera "dentada"	Frontera "suave"

Matemáticamente:

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n}$$

> *"Si cada árbol tiene 60% de accuracy, 100 árboles votando pueden tener 90%+"*

🚀 BLOQUE 4

Gradient Boosting

"Aprender de los errores del pasado"

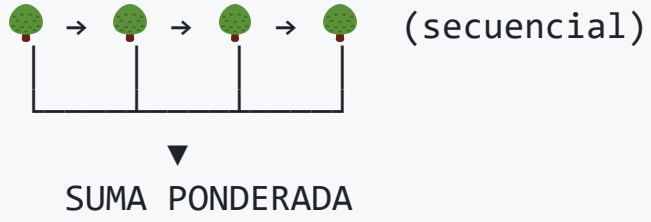
🚀 Boosting vs Bagging

Bagging (Random Forest)

- Árboles entrenados **en paralelo**
- Cada uno es **independiente**
- **Objetivo:** Reducir **varianza**
- Difícil de "romper"

Boosting (XGBoost/LightGBM)

- Árboles entrenados **en serie**
- Cada uno **corrige al anterior**
- **Objetivo:** Reducir **sesgo**
- Más preciso pero sensible



🚀 ¿Cómo Funciona Gradient Boosting?

El Algoritmo Paso a Paso

1. **Modelo Base:** Predice el promedio (o la moda)
2. **Calcular Residuos:** $r_i = y_i - \hat{y}_i$
3. **Entrenar Árbol 2:** Predice los residuos del Árbol 1
4. **Actualizar:** $\hat{y}_{nuevo} = \hat{y}_{anterior} + \eta \cdot \text{Árbol}_2$
5. **Repetir:** Hasta N iteraciones

La Magia del Learning Rate (η)

$$\hat{y} = F_0 + \eta \cdot h_1(x) + \eta \cdot h_2(x) + ... + \eta \cdot h_n(x)$$

> 💡 **Trade-off:** `learning_rate` bajo + más árboles = mejor generalización pero más lento

🏆 Los Campeones de Kaggle

Algoritmo	Creador	Fortaleza
XGBoost	Tianqi Chen	Robusto, bien documentado, regularización L1/L2
LightGBM	Microsoft	Velocidad, manejo nativo de categóricas
CatBoost	Yandex	Excelente con categóricas, menos tuning

En Código (LightGBM)

```
import lightgbm as lgb

lgbm = lgb.LGBMClassifier(
    n_estimators=200,
    max_depth=10,
    learning_rate=0.1,      # Tasa de aprendizaje
    num_leaves=31,          # Complejidad del árbol
    class_weight='balanced'
)
```

⚠️ Cuidado con el Overfitting en Boosting

Señales de Alerta

- AUC en train >> AUC en test
- Mejora continua en train, estancamiento en test
- `n_estimators` muy alto sin early stopping

Soluciones

```
# ✅ Early Stopping
lgbm.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    callbacks=[lgb.early_stopping(50)]  # Para si no mejora en 50 rounds
)

# ✅ Regularización
lgbm = lgb.LGBMClassifier(
    reg_alpha=0.1,    # L1 regularization
    reg_lambda=0.1    # L2 regularization
)
```

☕ BREAK

15 minutos

📐 BLOQUE 5

Los Clásicos Geométricos

SVM y KNN

📐 SVM: Support Vector Machines

La Idea

Encontrar el **hiperplano óptimo** que maximiza el **margen** entre clases.

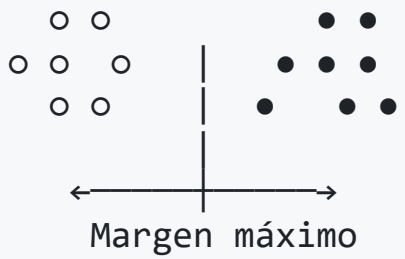
El Kernel Trick

- Proyecta datos a dimensiones superiores
- Donde SÍ son linealmente separables
- Sin calcular explícitamente la transformación

Kernels Comunes

- `linear` : Para datos linealmente separables
- `rbf` : Curvas suaves (más usado)
- `poly` : Fronteras polinomiales

Diagrama del Margen



En Código

```python
from sklearn.svm import SVC

svm = SVC(
    kernel='rbf',
    C=1.0,          # Regularización
    gamma='scale' # Alcance del kernel
)
```

⚠️ Limitaciones de SVM

Cuándo NO usar SVM

Situación	Problema
Datasets grandes (>50k filas)	Muy lento ($O(n^2)$ a $O(n^3)$)
Muchas features	Kernel RBF puede ser lento
Datos sin escalar	**SVM es muy sensible a la escala**
Necesitas probabilidades	Por defecto no las da bien

✅ Siempre Escalar Antes de SVM

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),  # ¡Obligatorio!
    ('svm', SVC(kernel='rbf'))
])
```

🔍 KNN: K-Nearest Neighbors

La Idea Más Simple

> *"Dime con quién andas y te diré quién eres"*

1. Encuentra los K vecinos más cercanos
2. Vota por la clase mayoritaria

Parámetros

- `n_neighbors`: Número de vecinos (K)
- `weights`: `uniform` o `distance`
- `metric`: `euclidean`, `manhattan`, etc.

Usos Modernos de KNN

Aplicación	Descripción
Imputación	`KNNImputer` para nulos
Recomendación	Usuarios similares
Detección de outliers	Distancia a vecinos

⚠️ Limitaciones

- **Lento** en predicción ($O(n)$ por cada query)
- Sufre la "maldición de la dimensionalidad"
- Sensible a features irrelevantes

🥊 BLOQUE 6

Arena de Combate

Random Forest vs LightGBM vs XGBoost

🥊 El Combate: Configuración

Dataset: Telco Churn (~7,000 clientes)

Objetivo: Predecir riesgo de impago de equipos

Los Contendientes

Modelo	Tipo	Configuración
🌲 **Random Forest**	Bagging	200 árboles, max_depth=10
⚡ **LightGBM**	Boosting	200 rounds, lr=0.1
🚀 **XGBoost**	Boosting	200 rounds, lr=0.1

Métricas de Evaluación

- **AUC-ROC:** Capacidad de discriminación
- **Tiempo:** Velocidad de entrenamiento

🏆 Resultados del Combate

Tabla de Resultados

Modelo	AUC	Tiempo
🌲 **Random Forest**	**0.9483** 🥇	**0.66s** 🥇
🚀 XGBoost	0.9466	0.96s
⚡ LightGBM	0.9457	3.48s

🎉 ¡Sorpresa! Random Forest Ganó

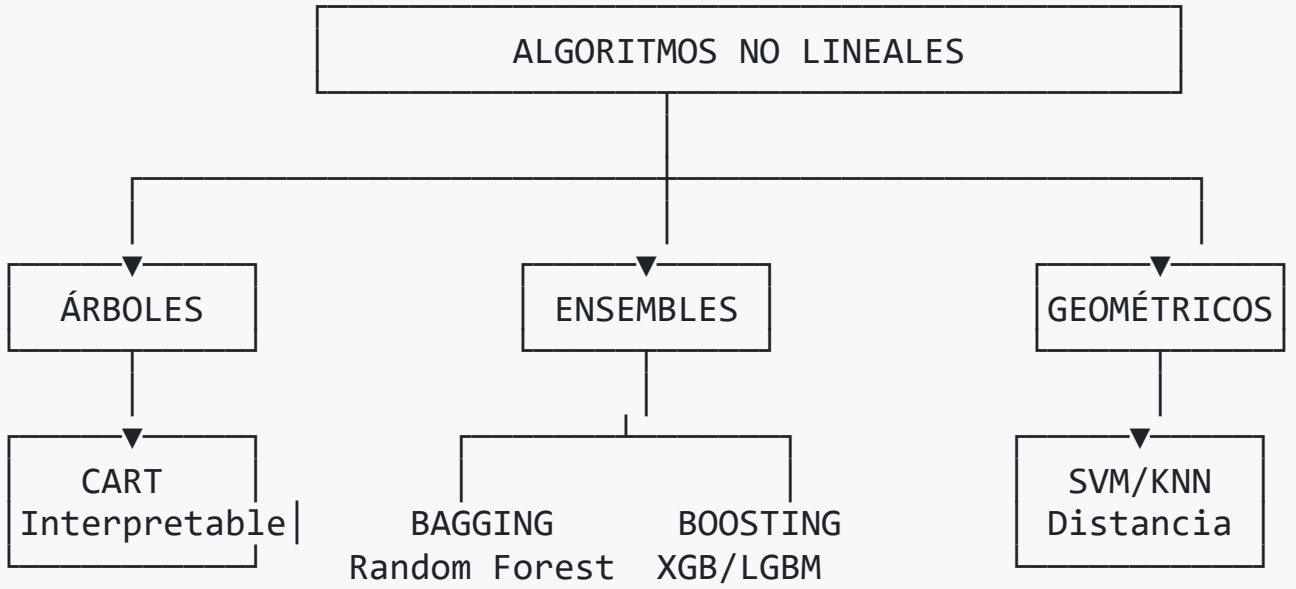
> **Lección:** Los benchmarks de internet no siempre aplican a TU dataset. En datasets pequeños/medianos, la paralelización de RF puede superar al boosting secuencial.

💡 ¿Cuándo Usar Cada Modelo?

Situación	Recomendación
Dataset pequeño/mediano (<50K)	🌲 **Random Forest**
Dataset grande (>100K filas)	⚡ **LightGBM**
Producción robusta, bien documentada	🚀 **XGBoost**
Baseline conservador	🌲 **Random Forest**
Competencias Kaggle	⚡🚀 **LightGBM/XGBoost** con tuning
Muchas categóricas	🐱 **CatBoost**

> 💡 **Pro-Tip:** Siempre prueba en TUS datos. No asumas que "el más nuevo es mejor".

📚 Resumen: Mapa de Algoritmos



🎯 Key Takeaways

1. **Linealidad tiene límites:** Usa algoritmos no lineales para patrones complejos
2. **Árboles solos = Overfitting:** Siempre controla `max_depth` y `min_samples_leaf`
3. **Bagging reduce varianza:** Random Forest es robusto y paralelo
4. **Boosting reduce sesgo:** XGBoost/LightGBM son más precisos pero sensibles
5. **No hay "mejor modelo universal":** Depende de tus datos, requisitos y recursos
6. **Siempre experimenta:** Los benchmarks no reemplazan la validación en TU dataset

📚 Referencias y Lecturas

Libros

- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.
- Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly.

Papers Fundamentales

- Breiman, L. (2001). *Random Forests*. Machine Learning, 45(1), 5-32.
- Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. KDD.
- Ke, G. et al. (2017). *LightGBM: A Highly Efficient Gradient Boosting*. NeurIPS.

Documentación

- Scikit-Learn User Guide
- XGBoost Documentation
- LightGBM Documentation

🙋 ¿Preguntas?

📧 jrodriguezm216@gmail.com

🌐 jordandataexpert.com

💻 github.com/JordanKingPeru

💻 ¡Vamos al Código!

Notebooks de Hoy:

1. `01_Algoritmos_No_Lineales.ipynb` - Visualización de fronteras
2. `02_Arena_Combate.ipynb` - RF vs LightGBM vs XGBoost

Abrir: `03Material/02_Sesion_Arboles_Ensembles/notebooks/`