SESIÓN 04

De la Caja Negra a la Realidad Productiva

Curso:

Machine Learning Supervisado

🚀 Machine Learning Supervisado

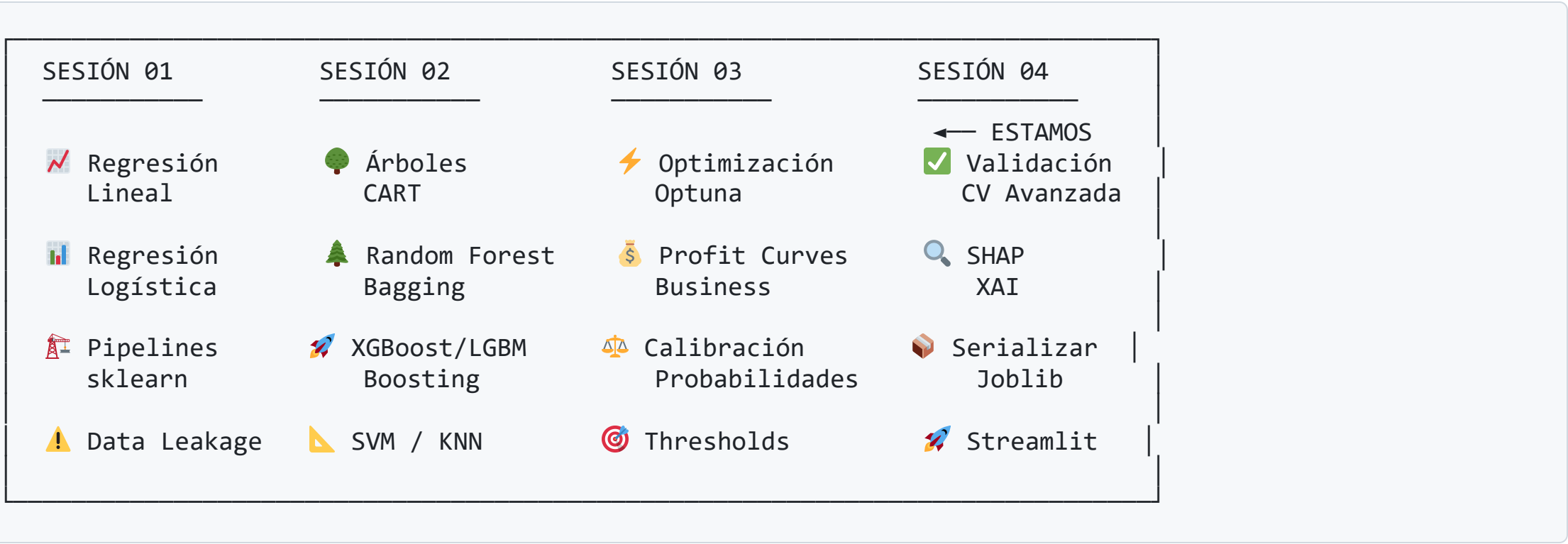
Sesión 04: De la Caja Negra a la Realidad Productiva

"El modelo más sofisticado no vale nada si no podemos explicarlo ni ponerlo en producción"

📋 Agenda de Hoy

Bloque	Tema	Duración
1	✅ Validación Avanzada: Cross-Validation Robusto	30 min
2	🔍 Explainable AI: SHAP Values	45 min
3	📦 Serialización: Múltiples Formatos	25 min
☕	**Break**	15 min
4	🚀 Deployment: De Notebook a Aplicación	35 min
5	💻 Hands-On: Construyendo la App	30 min

📅 Roadmap del Curso



✅ BLOQUE 1

Validación Avanzada

Más allá del `train_test_split`

🎯 El Problema con Validación Simple

❌ Lo que hacemos:

```
X_train, X_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)
```

Problemas:

- 📉 Una sola partición = alta varianza
- 📅 No respeta el tiempo
- ⚖️ Puede desbalancear clases
- 👥 Ignora agrupaciones

✅ Lo que deberíamos hacer:

```
cv = StratifiedKFold(
    n_splits=5,
    shuffle=True
)
scores = cross_val_score(
    model, X, y, cv=cv
)
```

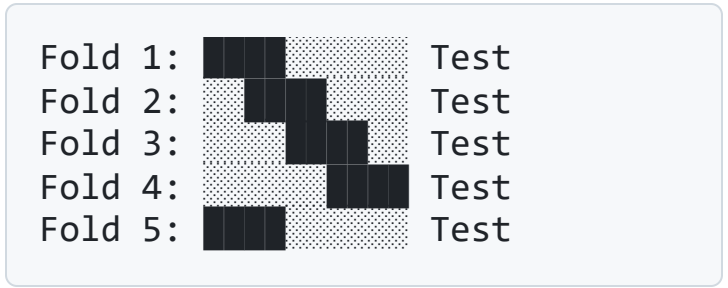
Beneficios:

- 📊 Múltiples evaluaciones
- 📈 Estimación de varianza
- 🎯 Métricas más robustas
- ✅ Menos overfitting oculto

🔄 Tipos de Cross-Validation

StratifiedKFold

Mantiene proporción de clases

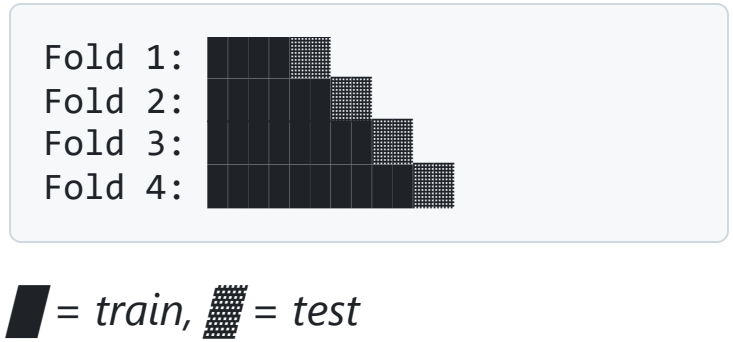


Usar cuando:

- Clasificación binaria
- Clases desbalanceadas

TimeSeriesSplit

Respeta orden temporal



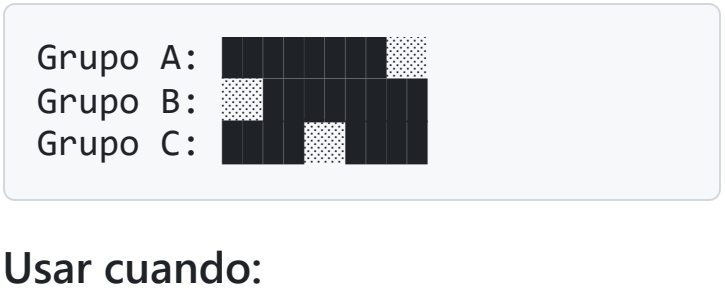
= train, = test

Usar cuando:

- Datos temporales
- Predicción futura

GroupKFold

Respeta agrupaciones



Usar cuando:

- Múltiples obs. por cliente
- Datos jerárquicos

⚠️ Data Leakage en Validación

🚨 ERROR CRÍTICO: Fuga de Información

El **futuro no puede predecir el pasado**. Si tu modelo "ve" datos futuros durante el entrenamiento, tus métricas serán engañosamente optimistas.

```
# ❌ INCORRECTO: El scaler "ve" todo el dataset
scaler.fit(X)  # Contamina con información del test
X_scaled = scaler.transform(X)
X_train, X_test = train_test_split(X_scaled)

# ✅ CORRECTO: Fit solo en train
X_train, X_test = train_test_split(X)
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

💡 Resultados de Validación Robusta

📊 Nuestro modelo LightGBM:

Métrica	Media CV	Desv. Std	Interpretación
AUC-ROC	0.9804	±0.0015	Excelente discriminación
Precision	0.9312	±0.0089	Alta precisión en positivos
Recall	0.8847	±0.0156	Buena captura de defaults
F1-Score	0.9073	±0.0102	Balance precision-recall

> 📖 **Regla de Oro:** Si la desviación estándar es muy alta (>0.05), tu modelo es inestable y probablemente está sobreajustando.

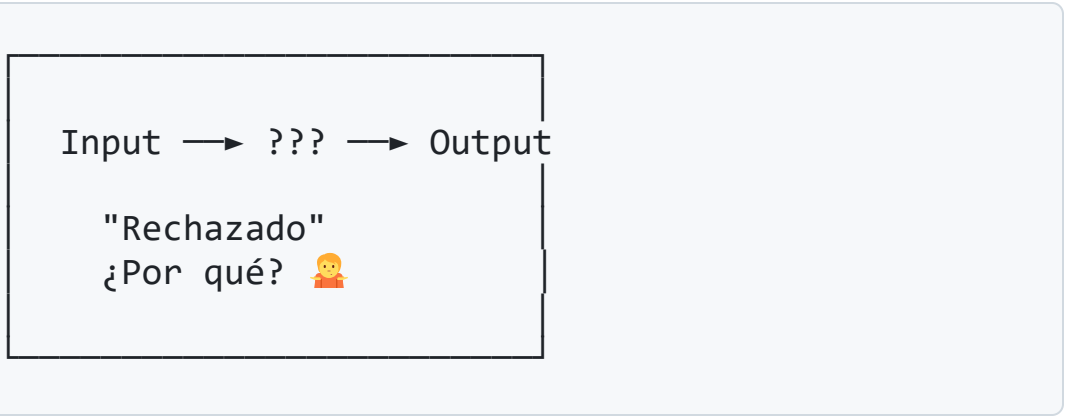
🔍 BLOQUE 2

Explainable AI (XAI)

SHAP: El Arte de Interpretar Cajas Negras

🤔 ¿Por qué Necesitamos XAI?

El Dilema de la Caja Negra



El negocio pregunta:

- ¿Por qué rechazamos a Juan?
- ¿Qué variable es más importante?
- ¿Es justo el modelo?

Regulaciones que lo exigen

- **GDPR (Europa):** "Right to explanation"
- **SBS (Perú):** Modelos de scoring explicables
- **Basilea III:** Documentación de riesgos

> ⚠️ **Riesgo Legal:** Un modelo que no puede explicar sus decisiones puede ser considerado discriminatorio.

🎮 SHAP: Teoría de Juegos para ML

¿Qué es un SHAP Value?

Es la **contribución marginal** de cada feature a la predicción, basada en el concepto de **Valores de Shapley** de teoría de juegos cooperativos.

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [f(S \cup \{i\}) - f(S)]$$

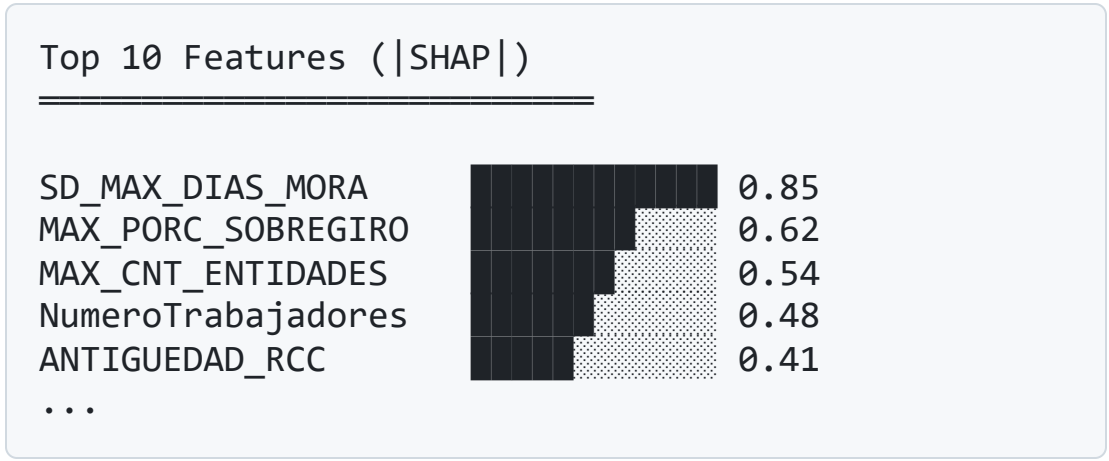
En palabras simples:

- Cada feature es un "jugador" en un juego
- El "premio" es la predicción final
- SHAP calcula cuánto aporta cada jugador al resultado

📊 Interpretación Global: Bar Plot

¿Qué muestra?

La **importancia promedio** de cada variable en el modelo completo.



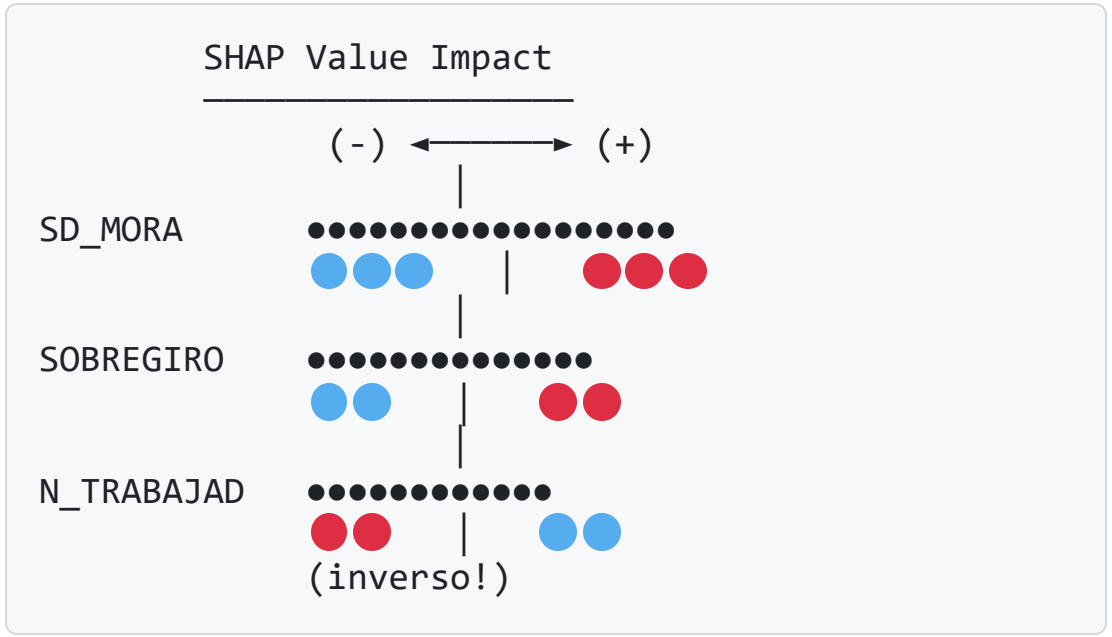
🔍 Insights del Modelo:

1. **SD_MAX_DIAS_MORA_SSFF_06M**
    - Variable más importante
    - Volatilidad en días de mora
2. **MAX_PORC_DEUDA_SOBREGIRO**
    - Uso agresivo de sobregiro
    - Señal de estrés financiero
3. **MAX_CNT_ENTIDADES_SSFF**
    - Diversificación de deuda

🐝 Interpretación Detallada: Beeswarm Plot

¿Qué muestra?

- **Posición X:** Impacto en predicción
- **Color:** Valor de la variable
  - 🔴 Rojo = Valor alto
  - 🔵 Azul = Valor bajo
- **Dispersión Y:** Distribución



🔍 Patrones Clave:

SD_MAX_DIAS_MORA:

- Valores altos (🔴) → Derecha (+)
- Indica mayor prob. de default

NumeroTrabajadores:

- Valores altos (🔴) → Izquierda (-)
- Empresas grandes = menor riesgo

🔥 Heatmap: Visión Matricial

> **Interpretación del Heatmap SHAP**
>
> El heatmap muestra los **SHAP values de cada observación** ordenadas por similitud. Permite identificar **clusters de comportamiento** y **patrones grupales**.

Columnas rojas intensas:

- Clientes con alto riesgo
- Contribuciones consistentemente positivas

Columnas azules:

- Clientes de bajo riesgo
- Contribuciones negativas

Filas más variables:

- Features discriminantes
- Alto poder predictivo

Filas uniformes:

- Features menos relevantes
- Poco impacto diferencial

🎯 Interpretación Local: Waterfall Plot

¿Qué muestra?

La **explicación individual** de una predicción específica.

```
Cliente ID: 12345
──────────────────────────
Predicción: 0.87 (Alto Riesgo)

Desglose:
┌──────────────────────────┐
│ Base Value    = 0.25     │
│ + MORA (+0.35)           │
│ + SOBREGIRO (+0.18)      │
│ + ENTIDADES (+0.12)      │
│ - TRABAJADORES (-0.08)   │
│ + OTROS (+0.05)          │
│ ═══════════════════════  │
│ Final         = 0.87     │
└──────────────────────────┘
```

💼 Uso en Negocio

Para el cliente:

> "Su solicitud fue rechazada principalmente debido a:
>
> 1. Historial de mora variable (35%)
> 2. Alto uso de sobregiro (18%)
> 3. Múltiples entidades financieras (12%)"

Para el analista:

> "Revisar si el cliente puede reducir exposición en alguna entidad"

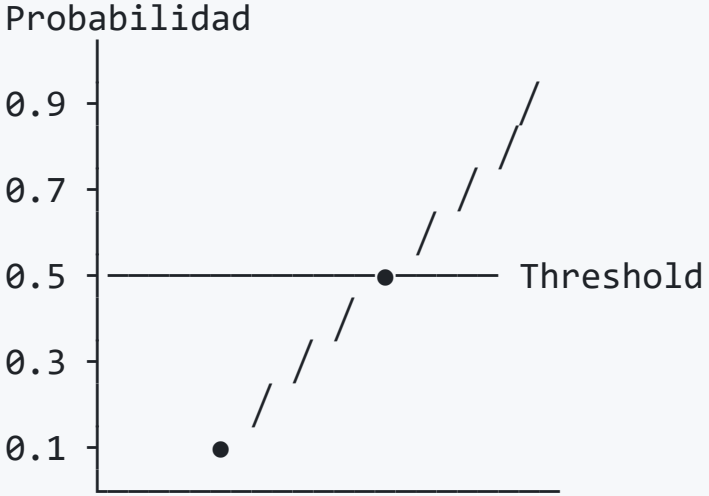
📈 Decision Plot: Trayectoria de Decisión

¿Qué muestra?

La **evolución acumulativa** del SHAP value desde el valor base hasta la predicción final.

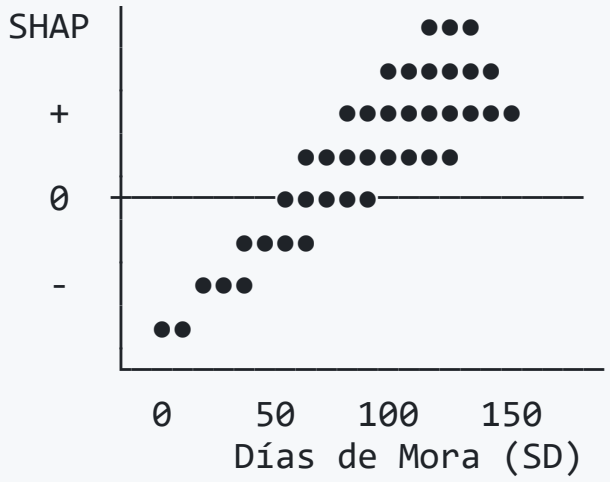
Interpretación:

- **Líneas que suben:** Mayor riesgo
- **Líneas que bajan:** Menor riesgo
- **Punto de quiebre:** Donde cruza el threshold



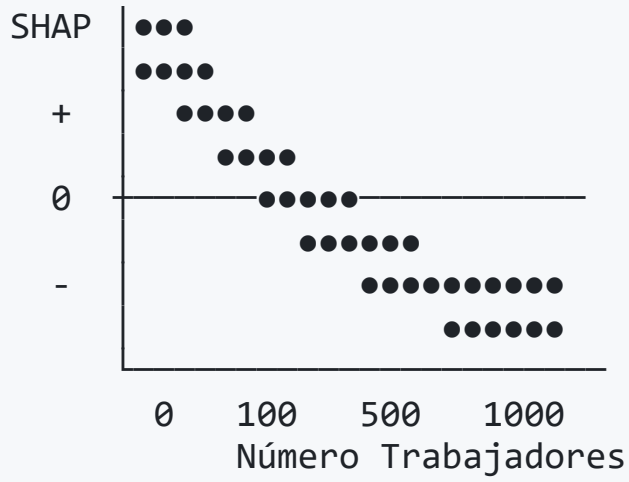
🔬 Scatter Plots: Relaciones No Lineales

SD_MAX_DIAS_MORA_SSFF_06M



Insight: Relación no lineal. Después de ~50 días, el impacto se estabiliza.

NumeroTrabajadores



Insight: Empresas grandes (<500 trabajadores) tienen menor riesgo crediticio.

📦 BLOQUE 3

Serialización de Modelos

Guardando el Conocimiento

💾 ¿Por qué Serializar?

El Problema

```
# Entrenas por horas...
model = lgb.LGBMClassifier()
model.fit(X_train, y_train)

# Cierras el notebook...
# 💀 Todo se pierde!
```

La Solución

```
# Guardar
joblib.dump(model, 'model.joblib')

# Meses después...
model = joblib.load('model.joblib')
model.predict(nuevo_cliente)  # ✅
```

Beneficios

Aspecto	Sin Serializar	Con Serializar
Tiempo	Re-entrenar	Cargar en ms
Reproducibilidad	❌	✅
Deployment	Imposible	Posible
Versiones	Ninguna	Control total

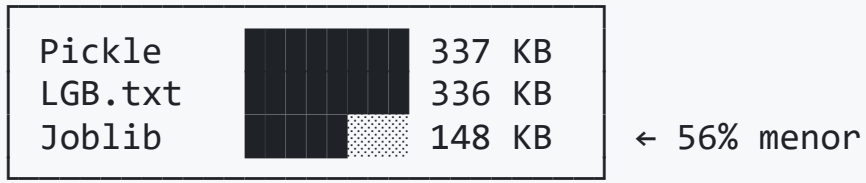
🛠️ Métodos de Serialización

Método	Formato	Tamaño	Cross-Platform	Uso Recomendado
Pickle	`.pkl`	Grande	Solo Python	Desarrollo rápido
Joblib	`.joblib`	**Comprimido**	Solo Python	**Producción Python**
LightGBM Native	`.txt`	Texto	Multi-lenguaje	Inspección manual
ONNX	`.onnx`	Optimizado	**Universal**	Edge, móvil
PMML	`.pmml`	XML	Enterprise	Java, SAS

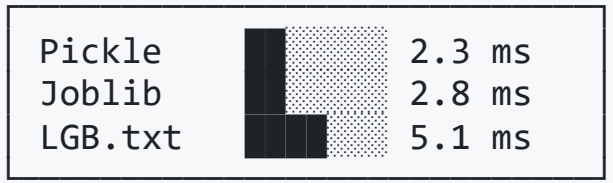
> 💡 **Recomendación:** Para producción en Python, usa **Joblib**. Para deployment multi-plataforma, considera **ONNX**.

📊 Comparativa de Formatos

Tamaño en Disco



Tiempo de Carga



Joblib con compress=3:

- Compresión zlib integrada
- Carga igual de rápida
- Ideal para deploy

Tiempos de inferencia:

- Todos similares (~0.1ms/pred)
- La diferencia es despreciable

📝 Metadatos: No Olvides el Contexto

```
metadata = {
    "model_name": "credit_scoring_lgbm_v1",
    "version": "1.0.0",
    "created_at": "2025-01-15T10:30:00",
    "author": "Jordan Rodriguez",

    # Información del entrenamiento
    "training_samples": 45231,
    "features": ["SD_MAX_DIAS_MORA", "MAX_PORC_SOBREGIRO", ...],
    "target": "default_flag",

    # Métricas
    "metrics": {
        "auc_roc": 0.9804,
        "threshold_optimal": 0.35
    },

    # Reproducibilidad
    "random_state": 42,
    "library_versions": {"lightgbm": "4.1.0", "sklearn": "1.3.0"}
}

with open("model_metadata.json", "w") as f:
    json.dump(metadata, f, indent=2)
```

🚀 BLOQUE 4

Deployment

Del Notebook a la Aplicación Web

🌐 Opciones de Deployment

🖥️ Streamlit

Aplicación web interactiva

Pros:

- ✅ Super rápido de crear
- ✅ Python puro
- ✅ Visualizaciones fáciles

Contras:

- ❌ No escala bien
- ❌ Single-threaded

Ideal para:

- Demos
- Prototipos
- Herramientas internas

🔗 FastAPI

API REST profesional

Pros:

- ✅ Alta performance
- ✅ Async/await
- ✅ Auto-documentación

Contras:

- ❌ Requiere frontend
- ❌ Más código

Ideal para:

- Microservicios
- Integración sistemas
- Producción real

☁️ Cloud ML

AWS/GCP/Azure

Pros:

- ✅ Escala infinita
- ✅ Managed service
- ✅ MLOps integrado

Contras:

- ❌ Costo $$
- ❌ Vendor lock-in

Ideal para:

- Enterprise
- Alto volumen
- SLA crítico

💻 Anatomía de una App Streamlit

```python
import streamlit as st
import joblib

# 1 Cargar modelo (una sola vez)
@st.cache_resource
def load_model():
    return joblib.load("model.joblib")

model = load_model()

# 2 Interfaz de usuario
st.title("🏦 Credit Scoring App")

col1, col2 = st.columns(2)
with col1:
    dias_mora = st.slider("Días de Mora (Máx)", 0, 180, 30)
    pct_sobregiro = st.slider("% Sobregiro", 0.0, 100.0, 25.0)
with col2:
    n_entidades = st.number_input("N° Entidades", 1, 20, 3)
    trabajadores = st.number_input("N° Trabajadores", 1, 10000, 50)

# 3 Predicción
if st.button("🔮 Calcular Score"):
    X_new = [[dias_mora, pct_sobregiro, n_entidades, trabajadores]]
    proba = model.predict_proba(X_new)[0][1]
    st.metric("Probabilidad de Default", f"{proba:.1%}")
```

🎨 Versión Completa vs Básica

`app.py` (~450 líneas)

Versión de producción

- ✅ CSS personalizado UNI
- ✅ Gráfico de gauge Plotly
- ✅ SHAP waterfall integrado
- ✅ Semáforo de riesgo
- ✅ Métricas del modelo
- ✅ Manejo de errores

Comando:

```
streamlit run app.py
```

`app_basic.py` (~80 líneas)

Versión mínima viable

- ✅ Solo lo esencial
- ✅ Sliders + predicción
- ✅ Progress bar simple
- ✅ Fácil de entender

Comando:

```
streamlit run app_basic.py
```

> 💡 **Tip:** Empieza con `app_basic.py` y agrega features gradualmente.

🔒 Consideraciones de Producción

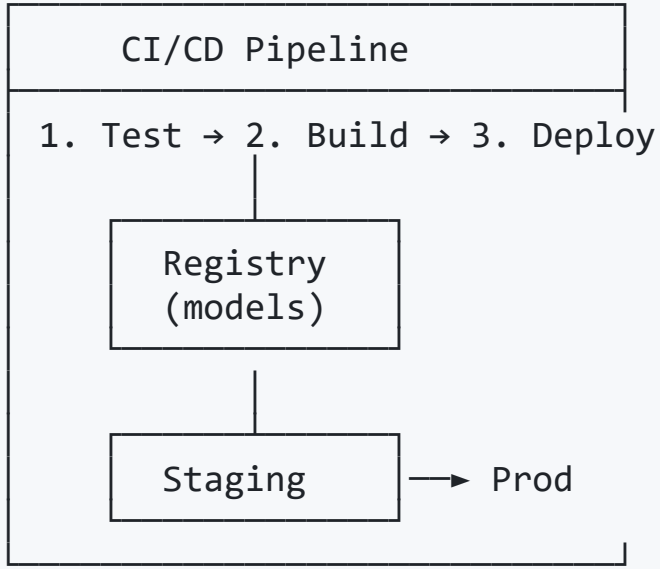
⚠️ Seguridad

- [ ] Validar inputs (rangos, tipos)
- [ ] Sanitizar strings
- [ ] Rate limiting
- [ ] Logging de predicciones
- [ ] Autenticación

📊 Monitoreo

- [ ] Drift detection (data/model)
- [ ] Latencia de predicción
- [ ] Tasa de errores
- [ ] Distribución de scores

🔄 MLOps



💻 BLOQUE 5

Hands-On

Construyendo Todo Junto

🛠️ Estructura del Proyecto

```
04_Sesion_XAI_Produccion/
├── notebooks/
│   ├── 01_Advanced_Validation.ipynb     # Cross-validation robusto
│   ├── 02_Explainable_AI_SHAP.ipynb     # SHAP completo
│   └── 03_Model_Serialization.ipynb     # Guardar modelos
│
├── app/
│   ├── models/
│   │   ├── model_joblib.joblib          # Modelo serializado
│   │   └── model_metadata.json          # Metadatos
│   ├── app.py                           # App completa
│   └── app_basic.py                     # App mínima
│
├── data/
│   └── dataset_creditscoring.csv        # Dataset
│
└── slides/
    └── S04_Presentacion.md              # Estos slides
```

🎯 Ejercicio Guiado

Paso 1: Ejecutar Notebooks (30 min)

```
01_Advanced_Validation.ipynb → Validación cruzada
02_Explainable_AI_SHAP.ipynb → Interpretación SHAP
03_Model_Serialization.ipynb → Guardar modelo
```

Paso 2: Probar la App (10 min)

```
cd app
streamlit run app_basic.py
```

Paso 3: Modificar y Experimentar (20 min)

- Agregar una variable más
- Cambiar el estilo del output
- Integrar un gráfico SHAP

🔑 Key Takeaways

1️⃣ Validación

> "Una métrica sin varianza es una mentira"

- Usar **K-Fold** siempre
- **Estratificado** para clasificación
- **Temporal** para series de tiempo

2️⃣ XAI

> "Si no puedes explicarlo, no lo despliegues"

- SHAP para interpretabilidad
- Global + Local
- Documentar para negocio

3️⃣ Serialización

> "El modelo más rápido es el que ya entrenaste"

- **Joblib** para Python
- **ONNX** para portabilidad
- Siempre guardar **metadatos**

4️⃣ Deployment

> "Un modelo en un notebook está muerto"

- Empezar simple (Streamlit)
- Escalar con FastAPI
- Automatizar con CI/CD

📚 Referencias y Lecturas

Papers Fundamentales

1. **Lundberg & Lee (2017)**
   "*A Unified Approach to Interpreting Model Predictions*"
   [NIPS 2017]

2. **Molnar (2022)**
   "*Interpretable Machine Learning*"
   [christophm.github.io/interpretable-ml-book]

3. **Sculley et al. (2015)**
   "*Hidden Technical Debt in ML Systems*"
   [Google Research]

Documentación Oficial

- 🟦 **SHAP:** shap.readthedocs.io
- 🟩 **Streamlit:** docs.streamlit.io
- 🟧 **Scikit-learn CV:** scikit-learn.org/stable/modules/cross_validation.html
- 🟥 **ONNX:** onnx.ai/sklearn-onnx

Cursos Recomendados

- Kaggle: "*Machine Learning Explainability*"
- MLU: "*Responsible AI*"

🙋 ¿Preguntas?

📧 jrodriguezm216@gmail.com

🌐 jordandataexpert.com

💻 github.com/JordanKingPeru

🎓 ¡Felicidades!

Has completado el curso de Machine Learning Supervisado

Sesión 01: Fundamentos y Pipelines
Sesión 02: Árboles y Ensambles
Sesión 03: Optimización y Calibración
Sesión 04: XAI y Producción ← ✅

> *"El verdadero aprendizaje comienza cuando aplicas esto en un proyecto real"*

💻 ¡Vamos al Código!

Abrir: `notebooks/01_Advanced_Validation.ipynb`

```
Flujo recomendado:
──────────────────
1 Validation    → Métricas robustas
2 SHAP          → Interpretación
3 Serialization → Guardar
4 App           → Desplegar
```