II PRÁCTICA APRENDIZAJE AUTOMÁTICO I

APRENDIZAJE SUPERVISADO: APROBACIÓN DE PRÉSTAMOS

Jorge López Díaz (jorge.lopez.diaz@alumnos.upm.es)

Mateo Bordoy Bardolet (mateo.bordoy@alumnos.upm.es)

11 - 12 - 2025

ÍNDICE

1. OBJETIVO DE LA PRÁCTICA

El propósito principal de este estudio es desarrollar, evaluar y comparar diferentes modelos de aprendizaje automático supervisado para abordar un problema de clasificación binaria en el sector financiero: la predicción de la aprobación de un préstamo. El sistema utiliza un conjunto de datos realista con información personal, laboral y financiera de solicitantes en Estados Unidos y Canadá.

El DataSet usado para la realización de esta práctica se puede encontrar en el siguiente enlace: https://www.kaggle.com/datasets/parthpatel2130/realistic-loan-approval-dataset-us-and-canada

1.1. Objetivo General

La meta central es construir un modelo predictivo capaz de determinar la variable objetivo *loan_status* (1: Préstamo Aprobado, 0: Préstamo Rechazado). La implementación exitosa de este modelo busca dotar a las instituciones financieras de herramientas para mejorar la toma de decisiones, optimizar la asignación de crédito y, fundamentalmente, reducir el riesgo de impago.

1.2. Objetivos Específicos y Estrategia Financiera

Dada la naturaleza del problema, el estudio no busca simplemente maximizar la tasa de acierto global (Accuracy), sino alinear el rendimiento del modelo con la rentabilidad y seguridad bancaria mediante los siguientes objetivos específicos:

Minimización del Riesgo Financiero (Falsos Positivos): Se establece como prioridad crítica la reducción de los Falsos Positivos, definidos como la aprobación incorrecta de un préstamo a un cliente que terminará en impago (coste monetario). Este error conlleva una pérdida directa de capital, siendo mucho más costoso que rechazar a un buen cliente (coste de oportunidad).

Optimización mediante Métrica F0.7: Para reflejar esta asimetría de costes, se utiliza la métrica F-Beta con beta=0.7 como criterio de optimización (refit) en la búsqueda de hiperparámetros. Esta métrica instruye a GridSearch a ponderar la Precision por encima del Recall (relación 2:1), favoreciendo una estrategia de aprobación conservadora.

Comparativa de Arquitecturas: Se evalúa el rendimiento de múltiples familias de algoritmos para determinar cuál captura mejor la complejidad de los datos. Se implementarán mediante la librería **Scikit-Learn** de Python.

- K-Nearest Neighbors (KNN) .
- Support Vector Machines (SVM) en variantes lineales y no lineales
- Decision Trees
- Regresión Logística

Interpretabilidad de Factores de Riesgo: Más allá de la predicción, el estudio busca identificar qué variables (como el ratio deuda/ingresos o el score crediticio) son determinantes para la concesión del crédito.

2. FASE I & II: PREPARACIÓN Y PREPROCESAMIENTO DE DATOS

Esta etapa consolida la ingesta, depuración y transformación de los datos para garantizar un modelado eficiente:

- **Muestreo y Limpieza:** Para agilizar los ciclos de experimentación, se realizó un submuestreo estratificado reduciendo el dataset original de 50.000 a **5.000 observaciones**. Se eliminaron identificadores (IDs) sin valor predictivo y se confirmó la ausencia de valores nulos.
- **Selección de Características:** Tras analizar las correlaciones, se eliminó la variable *loan_to_income_ratio* debido a su colinealidad redundante con *payment_to_income_ratio*, conservando esta última por aportar mayor información sobre la capacidad de pago mensual.
- **División Estratégica (Train-Test Split):** Se dividieron los datos en conjuntos de entrenamiento (80%) y prueba (20%) utilizando la estratificación (*stratify*). Esto asegura que la proporción original de la clase objetivo (aprobados vs. rechazados) se mantenga constante en ambos subconjuntos, algo crítico en problemas desbalanceados.
- **Pipeline de Transformación:** Se implementó un *ColumnTransformer* para aplicar tratamientos diferenciados según el tipo de variable, evitando la fuga de datos (Data Leakage):
  - → **Transformación Logarítmica + Estandarización:** Aplicada a variables con sesgo alto (*annual_income, savings_assets*) para normalizar su distribución.
  - → **StandardScaler:** Para el resto de variables numéricas continuas (*age, credit_score*, etc.).
  - → **One-Hot Encoding:** Para variables categóricas, eliminando la primera categoría para evitar multicolinealidad.

3. FASE III: APLICACIÓN DE MODELOS

3.1. K-Nearest Neighbors

3.1.1. Decisiones clave en la Arquitectura del Modelo

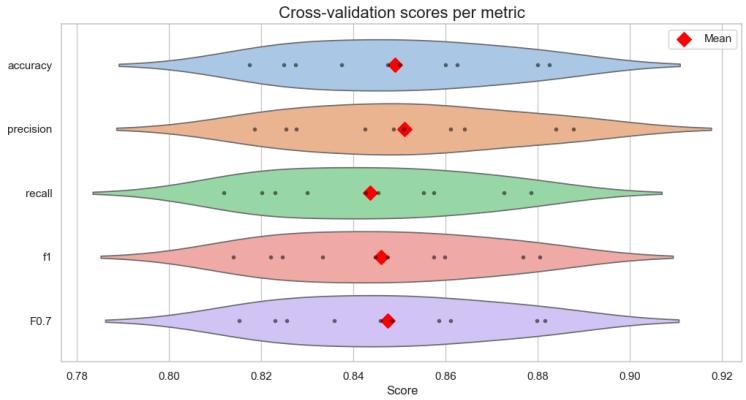
Pipeline y Prevención de Fugas: Se encapsuló el preprocesador dentro de un imbPipeline para evitar el Data Leakage. Esto garantiza que el escalado (StandardScaler) y la codificación se calculen independientemente en cada iteración de la validación cruzada.

Manejo del Desbalance (SMOTE): Dado que KNN no admite ponderación de clases (*class_weight*), se integró SMOTE en el pipeline. Esta técnica genera datos sintéticos de la clase minoritaria (impagos) para evitar que el modelo se sesgue hacia la clase mayoritaria por pura densidad.

Espacio de Búsqueda de Hiperparámetros:

- **Vecinos (n_neighbors):** Rango de 25 a 125 (impares). Se basó en la Regla de la Raíz Cuadrada $\sqrt{N}$, que sugiere un óptimo en torno a 60-65 vecinos para nuestro tamaño de entrenamiento. Se inicia en 25 para evitar el overfitting de valores bajos y se extiende hasta 125 para dar cabida a la estabilidad y la reducción de ruido.
- **Métrica de Distancia:** Se evalúa *Manhattan* junto a *Euclídea*. En espacios de alta dimensionalidad, la distancia Manhattan puede ser más robusta y ofrecer mejores resultados que la Euclídea.
- **Pesos (weights)**: Se introduce la ponderación por distancia frente a la uniforme. Esto permite comprobar si otorgar mayor influencia a los vecinos más cercanos (perfiles más similares) refina la frontera de decisión en casos ambiguos.

3.1.2. Resultados del modelo en cross-validation




```
Best params      :
{'knn__metric': 'euclidean',
'knn__n_neighbors': 67,
'knn__weights': 'uniform'}
```

3.2. SVM lineal

3.2.1. Decisiones clave en la Arquitectura del Modelo

Manejo del Desbalance: Pesos vs. SMOTE. A diferencia del pipeline de KNN, donde se utilizó SMOTE para generar datos sintéticos, en este caso se emplea *class_weight='balanced'*. SVM permite modificar directamente la función de pérdida asignando una penalización mayor a los errores en la clase minoritaria. Este enfoque es computacionalmente más eficiente y menos propenso a introducir ruido que el sobremuestreo.

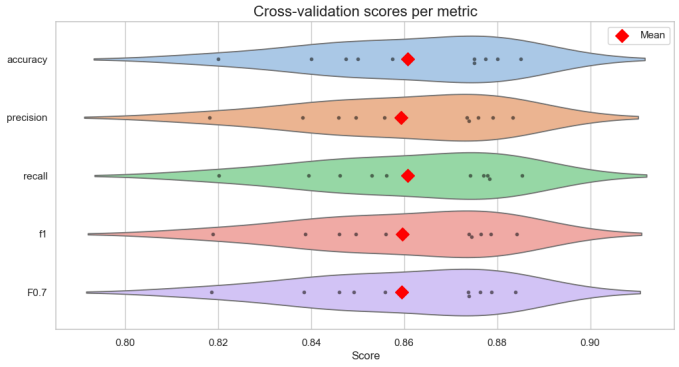
Configuración de Hiperparámetros.

- *C* (Regularización): Se pusieron valores en escala logarítmica desde 0.001 hasta 100. Controla el equilibrio entre simplicidad y precisión. Un valor bajo evita el sobreajuste, mientras que un valor alto hace al modelo más estricto con los errores de clasificación.
- *dual=False*: Se establece explícitamente porque en el conjunto de datos hay más muestras que características. Esto hace que el problema de optimización primario sea más rápido de resolver que el problema dual.

- *L1 / L2* (Penalización): Se prueban penalizaciones L1 (Lasso) para potencialmente anular características irrelevantes y L2 (Ridge) como método estándar de contracción de coeficientes.

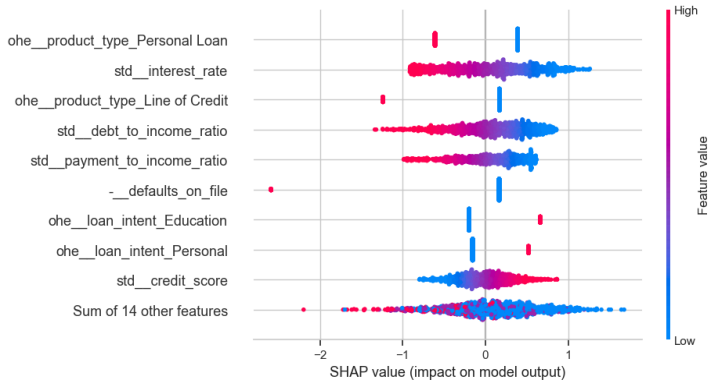
Validación Cruzada Estratificada: Se utiliza para asegurar que cada partición del proceso de validación cruzada mantenga la proporción original de clases, lo cual es crítico para que el parámetro class_weight calcule correctamente las frecuencias inversas.

3.2.2. Resultados del modelo en cross-validation




```
Best params: {'svm__C': 1, 'svm__class_weight': 'balanced', 'svm__dual': False, 'svm__loss': 'squared_hinge', 'svm__max_iter': 5000, 'svm__penalty': 'l2'}
```

Gráfica de valores SHAP para feature importance: Se introdujo una gráfica usando el LinearExplainer de SHAP para ver el comportamiento de las variables más importantes en el modelo. En el Jupyter Notebook se analiza con detalle.



3.3. SVM no lineal

Se decidió realizar un modelo no lineal aparte para poder compararlo con su versión lineal. Para ahorrar redundancias con las decisiones de arquitectura de este modelo (muy parecidas al SVM lineal) solo se explicarán aquellas que difieren.

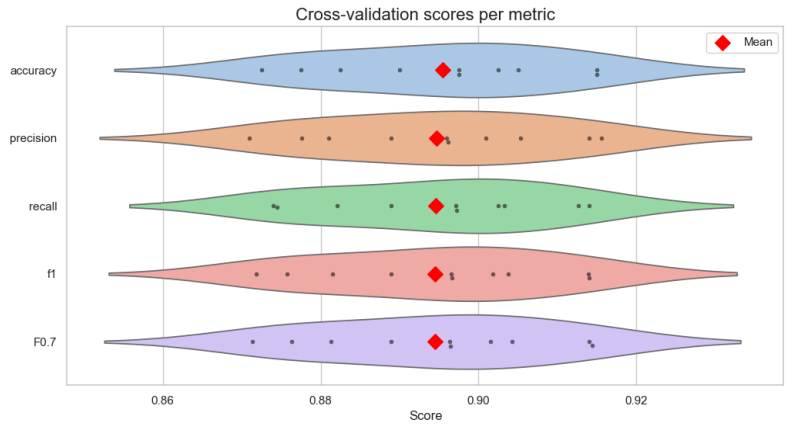
3.3.1. Decisiones clave en la Arquitectura del Modelo

Parámetro del Kernel: Esta es la mejora principal respecto al modelo anterior. Probamos *rbf* (Función de Base Radial), *poly* (Polinomial) y *sigmoid*. A diferencia del SVM lineal, estos kernels permiten que el modelo trace fronteras curvas o complejas para separar las clases objetivo.

Ajuste de la Curvatura:

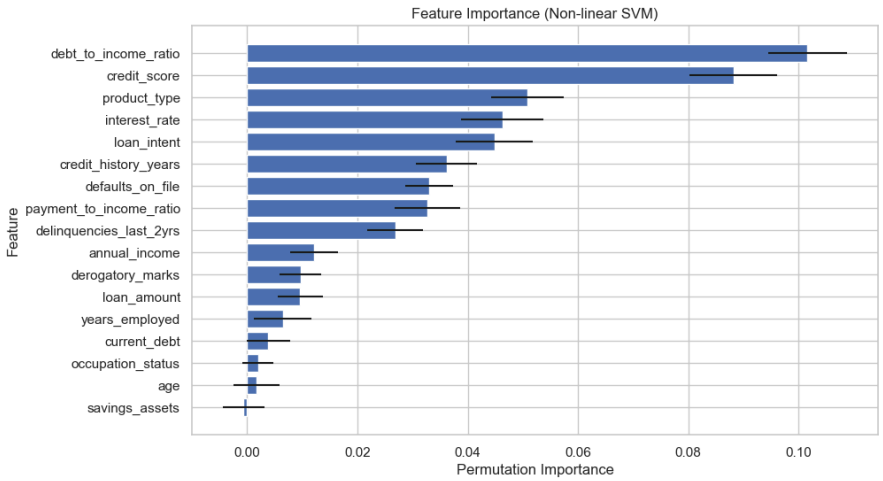
- *gamma*: Controla qué tan grande es el área de influencia de un punto de entrenamiento (es decir, hasta dónde se extiende el efecto de un punto en la frontera de decisión). Probamos scale y auto para comprobar si el modelo necesita un ajuste más "apretado" alrededor de los datos o una frontera más suave.
- *degree*: Usado específicamente para el kernel polinomial, nos permite comprobar si existe alguna relación cuadrática (2) o cúbica (3) entre las características

3.3.2. Comportamiento en cross-validation




```
Best params: {'svm__C': 10, 'svm__class_weight': 'balanced', 'svm__degree': 2, 'svm__gamma': 'auto', 'svm__kernel': 'rbf'}
```

Gráfica Permutation Importance: Dado que los SVM no lineales no tienen pesos de características inherentes, la Importancia por Permutación ofrece una métrica directa e intuitiva sobre cuánto se degrada la precisión del modelo cuando se elimina la información de una característica específica. Se evalúa en detalle en el Jupyter Notebook.



3.4. Árbol de Decisión

3.4.1. Decisiones clave en la Arquitectura del Modelo

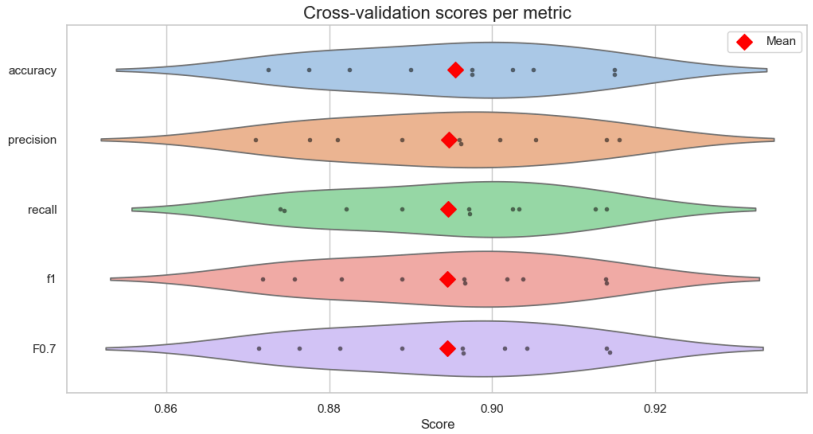
Regularización: Dado que los árboles de decisión tienden a memorizar los datos de entrenamiento, se limitó su complejidad para garantizar la generalización:

- Profundidad (*max_depth*): Se probaron valores entre 5 y 16. Valores bajos mejoran la interpretabilidad, mientras que None permite un crecimiento ilimitado (mayor riesgo de overfitting).
- Muestras Mínimas: Se ajustaron *min_samples_split* y *min_samples_leaf* para evitar que el modelo cree reglas demasiado específicas basadas en outliers o ruido.

Manejo del Desbalance (*class_weight='balanced'*): Se configuró el peso balanceado para ajustar el cálculo de impureza en las divisiones. Esto penaliza más los errores de clasificación en la clase minoritaria (impagos), forzando al árbol a crear ramas específicas para detectar estos casos críticos.

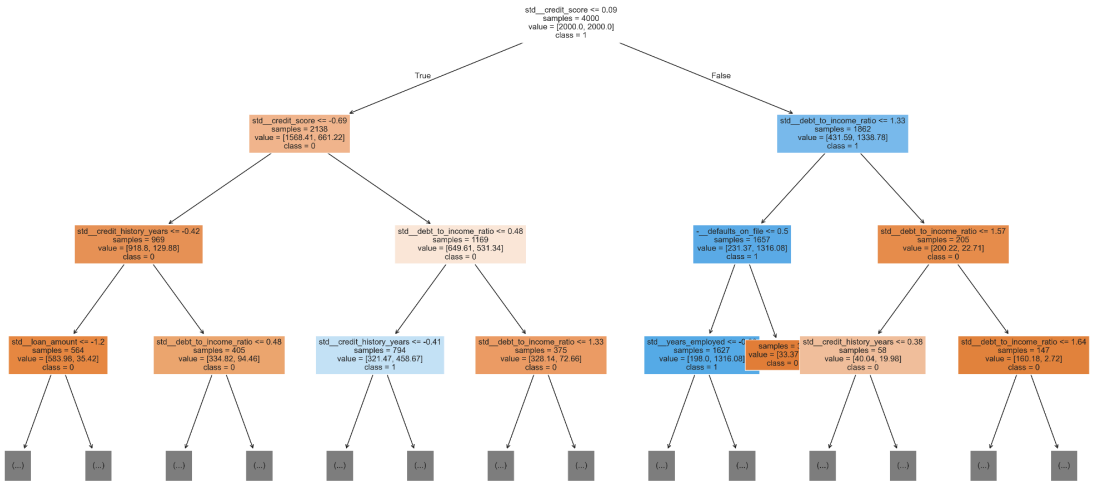
Aleatoriedad de Características (*max_features*): Se evaluaron opciones como sqrt y log2 frente a considerar todas las variables (None). Al buscar la mejor división usando solo un subconjunto aleatorio de características en cada nodo, se evita que el modelo dependa excesivamente de una variable dominante y se reduce el sobreajuste.

3.4.2. Comportamiento en cross-validation




```
Best params:
{'dt__class_weight':
'balanced', 'dt__criterion':
'gini', 'dt__max_depth': 8,
'dt__max_features': None,
'dt__min_samples_leaf': 10,
'dt__min_samples_split': 5}
```

3.4.3. Diagrama de Árbol e interpretación



Nota: Solo se muestran 3 niveles de profundidad, ya que los niveles superiores generan un diagrama poco claro. Se pueden ajustar figsize y max_depth para una visualización más detallada.

División Raíz (Discriminador Principal): *credit_score* ≤ 0.09 actúa como filtro primario. Separa a los solicitantes en una zona de Alto Riesgo (Rama Izquierda/Naranja) frente a una zona Segura (Rama Derecha/Azul).

Rama Izquierda (Zona de Riesgo): Ante scores bajos, el modelo verifica la severidad. Si el puntaje es crítico (≤ 0.69) y el historial crediticio es corto, el rechazo es casi absoluto (Naranja Oscuro). Sin embargo, un endeudamiento bajo respecto a las ganancias (*debt_to_income*) puede "rescatar" perfiles con puntajes mediocres, permitiendo su aprobación.

Rama Derecha (Zona Segura): A pesar de un buen score, el modelo busca "banderas rojas". Si el ratio de endeudamiento ante ganancia es alto (*debt_to_income* > *1.33*), el solicitante es reclasificado como riesgo (Naranja), demostrando que un buen crédito no compensa una deuda excesiva. Por el contrario, deuda baja y ausencia de impagos previos conducen a la clasificación de máxima seguridad (Azul Oscuro)

Conclusión: El árbol replica un razonamiento humano lógico: prioriza el puntaje crediticio y luego busca atenuantes (para perfiles bajos) o descalificadores (para perfiles altos). Pese a su excelente interpretabilidad, mostró un rendimiento inferior y un ligero mayor overfitting comparado con los modelos SVM.

3.5. Regresión Logística

3.5.1. Decisiones clave en la Arquitectura del Modelo

Regularización: *C = [0.01, 0.1, 1, 10]*. Se han seleccionado estos números para que no haya ni underfitting excesivo ni tampoco overfitting ya que el objetivo principal es un modelo que generalice.

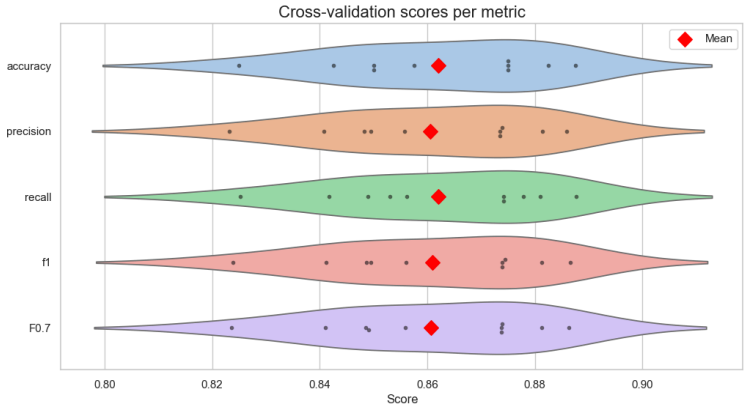
Solver='saga': Es el único algoritmo que soporta todos los tipos de penalización (L1, L2 y ElasticNet)

***penalty* = *['l1', 'l2', 'elasticnet']*:** Métodos de penalización propuestos.

- **L1** (Lasso): Puede llevar los coeficientes exactamente a cero, realizando efectivamente selección de características (eliminando variables inútiles).
- **L2** (Ridge): Reduce los coeficientes para evitar el sobreajuste pero mantiene todas las características.
- **ElasticNet**: Mezcla ambos métodos anteriores. El parámetro *l1_ratio=[0.5]* indica al modelo que los combine al 50/50.

Gráfica de valores SHAP para la importancia de cada característica:

3.5.2. Resultados en Cross-validation




```
Best params:  {'logreg__C':
10, 'logreg__class_weight':
'balanced',
'logreg__l1_ratio': 0.5,
'logreg__penalty': 'l2',
'logreg__solver': 'saga'}
```

4. SELECCIÓN DEL MEJOR MODELO

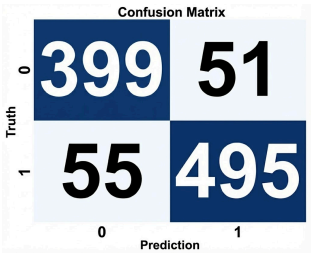
MÉTRICA (CV)	**NON-LINEAR SVM**	LOGISTIC REGRESSION	LINEAR SVM	DECISION TREE	KNN
F.07 SCORE	**0.8945**	0.8607	0.8595	0.8460	0.8475

El modelo seleccionado como el mejor es el **SVM con Kernel RBF.** Los motivos son los siguientes:

- Es el modelo con **F0.7 más alto en cross-validation**, con 0.8945 puntos. Además, el resto de métricas también están en torno a 0.89 (sacadas también en cross-validation). La gráfica de violín mostrada anteriormente resume perfectamente estos resultados.
- El modelo ha demostrado **generalizar competentemente**, ya que, aunque su evaluación en el conjunto train superó en métricas a la de test, la media no superó los 0.05 puntos lo que no nos permite rechazar el modelo por overfitting.

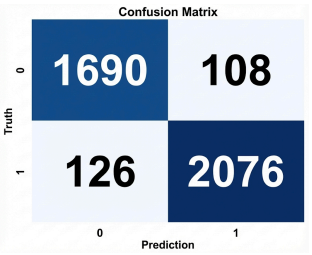
Se deja claro que la razón principal para seleccionar el modelo han sido los resultados obtenidos en cross-validation, dado que en situaciones realistas no se puede descartar o garantizar la validez de un modelo simplemente por su resultado en una predicción sobre el conjunto de test.

4.1. Resultados en la predicción: set *Test* y set *Train*



← Izq: Resultados *Test*

Dcha: Resultados *Train* →



El SVM no lineal alcanza el un rendimiento excepcional (89,4% de *accuracy*), confirmando la complejidad de los datos con una generalización estable. Su principal fortaleza radica en la gestión del riesgo: minimiza los Falsos Positivos a 51 (protegiendo el capital al detectar mejor a los morosos) y logra una precisión del 91% en aprobaciones. Simultáneamente, reduce el coste de oportunidad al disminuir los Falsos Negativos a 55, optimizando así la frontera de decisión para rechazar menos a buenos clientes y evitar préstamos tóxicos con mayor eficacia.

El *accuracy* en entrenamiento (0,942) supera a la de prueba (0,894), indicando un ligero sobreajuste; sin embargo, esta diferencia es aceptable dada la significativa mejora predictiva en datos nuevos (conjunto de *Test*).