

Proyecto Fase 1. Predicción del Riesgo Crediticio con Machine Learning

TC5044.10 – Operaciones de aprendizaje automático

Equipo 29

Data Engineer - Ingrid Pamela Ruiz Puga (A01021209)

Data Scientist – Fernando Emmanuel Perez Escalante (A01796934)

Data Engineer - Sebastián Cabezón Rosas (A01840261)

Software Enginer – Quirec Angeles Martínez (A01745050)

DevOps – Daniel Ivan Benitez Fernandez de Jauregui (A01795860)



ESCUELA DE INGENIERÍA Y CIENCIAS

MAESTRÍA EN INTELIGENCIA ARTIFICIAL APLICADA

12 de octubre de 2025

Introducción

Este documento presenta la Fase 1 del proyecto de predicción de riesgo crediticio, desarrollado por el equipo 29 de la Maestría en Inteligencia Artificial Aplicada del Tecnológico de Monterrey. El objetivo es implementar una solución basada en machine learning que permita clasificar solicitantes de crédito según su nivel de riesgo, utilizando herramientas y prácticas de MLOps para garantizar reproducibilidad, trazabilidad y calidad en el desarrollo.

El análisis se basa en el conjunto de datos South German Credit Update, el cual contiene información financiera y demográfica de mil solicitantes de crédito en Alemania. Este dataset incluye variables como historial crediticio, empleo, edad, ahorros y estado civil, lo que lo hace adecuado para problemas de clasificación binaria enfocados en scoring crediticio. A partir de estos datos, se realizaron tareas de limpieza, transformación y modelado, documentando cada etapa del proceso con base en buenas prácticas de ciencia de datos.

1. Resumen del Proyecto y ML Canvas

El dataset South German Credit contiene 1,000 ejemplos (700 créditos buenos, 300 malos) con 20 variables predictoras más la variable objetivo ('credit_risk': bueno/malo). Es un problema clásico de clasificación binaria orientado a predecir si un solicitante de crédito será buen o mal pagador, con aplicaciones directas en el ámbito financiero.

1.1 Naturaleza del problema

El objetivo es construir un modelo de clasificación binaria que prediga la probabilidad de impago de un solicitante. Esto permite a una entidad financiera tomar decisiones más informadas sobre aprobación o rechazo de crédito, reduciendo pérdidas por morosidad.

1.2 Variables principales

El dataset contiene variables relacionadas con historial crediticio, empleo, ahorros, propiedades, estado civil, edad y más. No hay valores faltantes y las variables están discretizadas o codificadas.

1.3 Principales retos, sesgos y riesgos

- Desbalance entre clases reales y dataset.
- Posibles sesgos demográficos (e.g., trabajador extranjero).
- Datos antiguos (años 70s) que limitan la vigencia.
- Necesidad de explicabilidad en modelos financieros.
- Costos de error asimétricos (falsos negativos más costosos).
- Riesgo de multicolinealidad entre variables.

1.4 Oportunidades

El dataset es ideal para experimentación, prototipado y benchmarking en proyectos de scoring crediticio. Permite comparar distintos algoritmos y técnicas de ingeniería de características,

El ML Canvas describe de forma estructurada los elementos de un proyecto de machine learning: objetivo, fuentes de datos, características, predicción, decisiones y evaluación. A continuación, se presenta la versión adaptada para el dataset South German Credit.

Machine Learning Canvas - South German Credit

- Objetivo / Propuesta de valor:

Construir un modelo de scoring de riesgo crediticio para clasificar solicitantes entre riesgo bajo y alto, reduciendo pérdidas y aumentando eficiencia en otorgamientos.

- Fuentes de datos:

Datos históricos de créditos, historial de pagos, burós de crédito, datos macroeconómicos.

- Recolección de datos:

Información obtenida al solicitar un crédito, más registros internos y externos de comportamiento de pago.

- Características (feature engineering):

Variables base del dataset más variables derivadas (ratios, transformaciones, interacciones).

- Construcción / actualización de modelos:

Entrenamiento inicial con datos históricos y reentrenamiento periódico con nuevos datos.

- Tarea ML / Predicción:

Clasificación binaria (default sí/no) usando regresión logística, árboles, ensemble o redes neuronales.

- Decisiones:

Usar la probabilidad estimada para aprobar, rechazar o ajustar condiciones del crédito.

- Momento de predicción:

En el momento de la solicitud o para monitorear cartera existente.

- Evaluación offline (validación):

Métricas: AUC-ROC, sensibilidad, especificidad, matriz de costos.

- Evaluación en producción / monitoreo:

Comparar predicciones con defaults reales, detectar drift y degradación del modelo.

- Feedback / retroalimentación:

Incorporar nuevos resultados de pago como etiquetas para reentrenar periódicamente.

- Limitaciones / riesgos / supuestos:

Sesgos de datos, falta de representatividad, cambios regulatorios, posibles variables discriminatorias.

Propuesta de valor del proyecto

La implementación de un modelo de scoring crediticio basado en aprendizaje automático ofrece valor tangible a una institución financiera al mejorar la toma de decisiones, reducir pérdidas, aumentar la rentabilidad y permitir procesos más ágiles y auditables.

Propuesta de valor general

- Incrementar la tasa de aprobación segura de créditos.
- Reducir pérdidas por impagos.
- Automatizar la decisión crediticia.
- Mejorar la transparencia mediante modelos explicables.
- Adaptación continua con datos actualizados.

Indicadores de éxito (KPI)

- Reducción en tasa de impago.
- Mejora en AUC/ROC del modelo.
- Aumento de créditos aprobados sin elevar riesgo.
- Ahorro operativo.
- ROI positivo del proyecto.

Riesgos asociados

- Falta de generalización del modelo.
- Variables no disponibles en producción.
- Cambios regulatorios.
- Resistencia organizacional al cambio.

Pipeline técnico

El proyecto se desarrolló siguiendo las buenas prácticas de MLOps, asegurando reproducibilidad, trazabilidad y automatización de todo el flujo.

Etapas del pipeline:

- Ingesta y limpieza de datos.
- Feature engineering (transformación y selección de variables relevantes).
- Entrenamiento y validación de modelos.
- Versionado de datos y modelos con DVC.
- Sincronización con Git y Dagshub.

Decisions	ML Task	Value Propositions.	Data Sources	Collecting Data
Utilizar la probabilidad estimada para aprobar, rechazar o ajustar las condiciones del crédito (tasa, monto, plazo).	Clasificación binaria (default sí/no) mediante algoritmos como regresión logística, árboles de decisión, ensambles o redes neuronales.	Construir un modelo de scoring de riesgo crediticio para clasificar solicitantes entre riesgo bajo y alto, reduciendo pérdidas y aumentando la eficiencia en los otorgamientos.	Datos históricos de créditos, historial de pagos, información de burós de crédito y datos macroeconómicos	Información obtenida al solicitar un crédito, combinada con registros internos y fuentes externas sobre comportamiento de pago.
Making Predictions	Offline Evaluation	Features	Building Models	
En el momento de la solicitud del crédito o durante el monitoreo de la cartera vigente.	Métricas: AUC-ROC, sensibilidad, especificidad y matriz de costos para evaluar el balance entre riesgo y retorno.	Uso de variables base del dataset junto con variables derivadas (ratios financieros, transformaciones, interacciones).	Entrenamiento inicial con datos históricos y reentrenamiento periódico conforme llegan nuevos datos.	
Live Evaluation and Monitoring				
Métricas: AUC-ROC, sensibilidad, especificidad y matriz de costos para evaluar el balance entre riesgo y retorno.				

Ver canvas en

https://github.com/secabezón/MLOps/blob/main/docs/MLCanvas_South_German_Credit.docx

Organización del Proyecto (Cookiecutter)

Estructura propuesta para la elaboración del presente proyecto:

```

├── src/          (scripts de limpieza/prep y modelado)
    ├── data/
        ├── raw/      (datos originales, gestionados por DVC)
        └── processed/ (datos limpios procesados con EDA)
            ├── models/   (artefactos de modelo, versionados con DVC)
            └── artifacts/ (modelos entrenados)
    ├── notebooks/  (EDA y experimentación)
    ├── params.yaml (config de prep/model/seed)
    ├── dvc.yaml (opcional) (pipeline declarativo)
    ├── .gitignore   (no subir .venv ni datos reales)
    └── README.md    (guía de uso)

```

Ver repositorio

EDA y Preprocesamiento

Este documento resume todas las modificaciones realizadas al dataset *German Credit* durante el EDA y la preparación para modelado. Incluye la configuración de versionado con DVC (usando Dagshub como remoto), las transformaciones aplicadas, motivos técnicos y enlaces a los artefactos generados.

Configuración DVC y Dagshub

Notas principales sobre cómo se configuró DVC para este proyecto (no incluye tokens/secretos):

- Preparar entorno y dependencias:
 - Activar el entorno virtual (ej.: .venv\Scripts\activate en Windows PowerShell).
 - Instalar DVC con soporte para Google Drive si se va a usar: pip install "dvc[gdrive]".
 - Verificar la instalación: dvc --version.
- Conectar con Dagshub como remoto DVC:
 - Crear/usar un remote DVC apuntando al repositorio Dagshub del proyecto.
Ejemplo:
 - dvc remote add -d dagshub <https://dagshub.com/Pamela-ruiz9/MLOps.dvc>
 - Autenticación/credenciales: usa el método seguro que prefieras (variables de entorno, login interactivo de DVC, o las integraciones de Dagshub). **No** guardar tokens en archivos de texto del repositorio.
 - Subir datos y metadatos con DVC:
 - dvc add <ruta_al_dataset> para trackear archivos grandes.
 - git add <.dvc files> && git commit -m "track data with dvc".
 - dvc push para enviar los datos al remote.
 - Finalmente, empujar cambios Git a Dagshub/remote git: git push dagshub main (o con --force-with-lease si es necesario, pero con cuidado).
- En este proyecto se registraron instrucciones y un ejemplo de comandos en un archivo local (suministrado por el equipo). Si vas a reproducir, reemplaza <usuario>/<repo> por los tuyos y asegúrate de usar un método de autenticación seguro.

Cambios aplicados al dataset

A continuación se documentan, por pasos lógicos, las transformaciones realizadas, con la razón técnica para cada una y los artefactos que las registran.

1. Normalización de nombres de columnas
- Qué se hizo:
 - Eliminación de espacios, conversión a minúsculas y reemplazo de espacios por guiones bajos (ej.: df.columns = df.columns.str.strip().str.replace(' ', '_').str.lower()).

- Por qué:
 - Estándarizar nombres evita errores posteriores (accesos por clave), mejora la reproducibilidad y facilita la escritura de scripts y pipelines.
 - Artefactos:
 - Cambio en los notebooks; reflejado indirectamente en los CSV resultantes.
2. Eliminación de columnas extras y columnas mixed-type
- Qué se hizo:
 - Se detectaron y eliminaron columnas que no existían en df_orig (posibles columnas añadidas accidentalmente) y columnas de tipo object con mezcla de valores numéricos y textuales que causarían problemas en pasos numéricos.
 - Por qué:
 - Columnas extra pueden venir de salidas parciales o metadatos inconsistentes. Las columnas mixed-type pueden romper conversiones numéricas o modelos (strings en columnas numéricas). Es preferible auditar y eliminar o aislar estas columnas antes del preprocesado.
 - Artefactos:
 - Registro en data/processed/cleaning_rules.json con anotaciones dropped_mixed_type.
3. Normalización y reemplazo de tokens de missing
- Qué se hizo:
 - Se identificaron tokens comunes que representaban missing (n/a, na, null, ?, unknown, error, invalid, none, '') y se reemplazaron por np.nan.
 - Limpieza de strings (strip, normalización de espacios, eliminación de caracteres invisibles).
 - Por qué:
 - En datasets reales, hay múltiples representaciones de missing; estandarizarlas permite conteos correctos, imputaciones consistentes y evita convertir estas cadenas en categorías útiles erroneamente.
 - Artefactos:
 - data/processed/cleaning_rules.json registra que se aplicó str_strip_and_normalize y replace_common_missing_tokens por columna.
4. Coerción de columnas numéricas (coerce to numeric)
- Qué se hizo:
 - Para columnas que df_orig marcaba como numéricas, se intentó forzar a numérico con pd.to_numeric(errors='coerce'). Valores no convertibles pasaron a NaN.
 - Por qué:
 - Garantizar tipo numérico es necesario para análisis estadístico, imputación y modelado. Coerción con errors='coerce' es segura porque registra en NaN las celdas no-convertibles para revisión posterior.
 - Artefactos:
 - rows_with_problematic_numeric_values_sample.csv (si se detectaron) y cleaning_rules.json con conteos antes/después.
5. Identificación de filas extra (multiplicity-aware)
- Qué se hizo:

- Se compararon df y df_orig de manera awareness de multiplicidad (conteo por fila/valores) para detectar las 20 filas extra que no pertenecen al df_orig.
 - Se exportaron estas filas a data/raw/extrarows_identified.csv para auditoría.
 - Por qué:
 - Mantener trazabilidad: no eliminar silenciosamente datos sin registrar. Identificar filas extra permite discutir si deben borrarse o corregirse.
 - Artefactos:
 - data/raw/extrarows_identified.csv (si se ejecutó la celda). Registro en cleaning_rules.json con anotación sobre filas extra.
6. Reemplazo de imputación previa y uso de KNNImputer (decisión del usuario)
- Qué se hizo:
 - Se implementó primero una opción de imputación median/mode, pero el usuario pidió explícitamente usar sólo KNNImputer. Se dejó la celda final que aplica KNNImputer(n_neighbors=5) sólo a columnas numéricas.
 - Si alguna columna numérica era completamente NaN, se rellenó temporalmente con mediana de df_orig (o 0) para que KNN pueda ejecutarse, y luego se registró la operación.
 - Por qué:
 - KNNImputer aprovecha la estructura multivariada para imputar valores plausibles. El usuario solicitó KNN por razones de robustez en este caso.
 - Artefactos:
 - data/raw/german_credit_clean_v1_knn_imputed.csv (nombre de ejemplo generado por notebook) y cleaning_rules.json con imputation info.
7. Detección y capping de outliers (IQR)
- Qué se hizo:
 - Se detectaron outliers usando el criterio IQR (k=1.5) por columna numérica.
 - Se aplicó capping: valores por debajo del límite inferior se reemplazaron por el mínimo no outlier; valores por encima del límite superior se reemplazaron por el máximo no outlier.
 - Se creó una copia df_capped y se guardó como CSV (*_capped.csv).
 - Por qué:
 - El capping reduce el efecto de outliers extremos en modelos sensibles y en estadísticas. Se eligió capping por límite no-outlier para mantener valores realistas en el rango observado.
 - Artefactos:
 - data/raw/german_credit_clean_v1_capped.csv (ejemplo), reports/figures/* con histogramas/boxplots y cleaning_rules.json con límite y conteos de outliers por columna.
8. Conversión a Int64 (dtype nullable)
- Qué se hizo:
 - Se ejecutó un cast directo a Int64 (dtype nullable de pandas) para columnas donde había valores numéricos detectables. Fracciones se redondearon a entero antes de convertir.
 - Por qué:
 - El user pidió explícitamente que las columnas estén en un formato entero. Int64 (nullable) permite representar NA junto a enteros sin forzar a float.

- Riesgo / Nota:
 - No todas las columnas categóricas deben convertirse a enteros sin una codificación explícita. Se aplicó cast solo cuando había valores numéricos detectables.
 - Artefactos:
 - Cambios en df en memoria y guardados posteriores; registro en cleaning_rules.json.
9. Guardado de reglas y artefactos de auditoría
- Qué se hizo:
 - Se guardó un archivo data/processed/cleaning_rules.json con un historial de las acciones por columna (coerciones, imputaciones, capping, casts, drops), además de muestras problemáticas.
 - Se generaron CSVs y figuras a reports/ y data/raw/ para revisión manual.
 - Por qué:
 - Trazabilidad y reproducibilidad: es crítico poder decir qué se hizo y por qué, y recuperar las filas problemáticas si es necesario.
10. Preparación para modelado
- Qué se hizo:
 - Se creó el notebook notebooks/modeling_and_training.ipynb que:
 - Fija target_col = 'kredit'.
 - Convierte las columnas categóricas (según la documentación suministrada) a category.
 - Construye un ColumnTransformer que aplica imputación + escalado a numéricas y OneHot/Ordinal a categóricas según cardinalidad.
 - Aplica SelectKBest si hay target para seleccionar un subconjunto de features.
 - Entrena dos modelos de referencia: LogisticRegression y RandomForest, y salva ambos pipelines (pipeline_logreg.joblib, pipeline_model.joblib).
 - Por qué:
 - Preparar un pipeline reproducible que incluya transformación y modelo en un solo objeto facilita despliegue y evaluación.
 - Artefactos:
 - models/pipeline_logreg.joblib, models/pipeline_model.joblib, reportes de métricas en la salida del notebook.

Versionado de Datos y Artefactos (DVC + DagsHub)

Una vez generados los datos limpios realizamos los siguientes pasos:

Flujo estándar (PowerShell/Windows):

```
# añadir raw y processed
dvc add src\data\raw\german_credit_modified.csv
dvc add src\data\processed\clean.csv
```

```
# subir punteros y config (NO los binarios reales)
```

```

git add data\processed\clean.csv.dvc models.dvc .dvc\config .gitignore
git commit -m "Track: clean y modelos con DVC"
git push origin main
git push dagshub main # opcional, para ver también el código en DagsHub

# subir blobs reales al remoto DVC (DagsHub)
dvc push

# verificar
dvc status -c

```

Reproducción para el equipo:

```

git pull origin main
pip install dvc
# configurar user/token de DagsHub (local)
dvc pull
# si hay rutas cambiadas:
dvc checkout

```

Modelo Base y Métricas

Después del preprocesamiento, se aplicó una selección de características (features) mediante el método estadístico ANOVA F-test, seleccionando las 20 variables más relevantes con el fin de reducir el ruido y mejorar el rendimiento del modelo.

Finalmente, se entrenaron y evaluaron tres modelos supervisados:

1. Random Forest

El Random Forest es un modelo basado en múltiples árboles de decisión independientes. El resultado se obtiene mediante el promedio de las predicciones de todos los árboles. Se eligió este modelo porque, al combinar varios árboles de decisión, reduce el sobreajuste (overfitting) y es capaz de capturar relaciones complejas entre las variables.

Parámetros utilizados:

- model = RandomForestClassifier: define la clase del modelo.
- n_estimators = 100: establece que se construirán 100 árboles de decisión. Este número ofrece un buen equilibrio entre precisión, rendimiento y uso de memoria.
- random_state = 42: permite reproducir los resultados del modelo.

2. Regresión Logística

La Regresión Logística es un algoritmo de clasificación lineal que modela la probabilidad de pertenencia a una clase mediante una función logística o sigmoide. Este modelo es simple, rápido de implementar y suele ajustarse bien a problemas de clasificación binaria.

Parámetros utilizados

- `max_iter = 1000`: establece el límite máximo de iteraciones que el modelo realiza para encontrar la solución óptima.
- `solver = 'lbfgs'`: especifica el método de optimización empleado para minimizar el error y encontrar los mejores parámetros durante el entrenamiento.

3. XGBoost

XGBoost (Extreme Gradient Boosting) es un método que construye árboles de decisión de forma secuencial, corrigiendo los errores de los árboles anteriores. Este modelo se destaca por ofrecer alta precisión y excelente rendimiento, aunque requiere mayor uso de recursos computacionales y un ajuste más fino de sus parámetros.

Parámetros utilizados:

- `n_estimators = 300`: número total de árboles que se construirán.
- `learning_rate = 0.1`: tasa de aprendizaje. Un valor pequeño permite que el modelo aprenda de forma gradual, reduciendo el riesgo de sobreajuste.
- `max_depth = 6`: profundidad máxima de cada árbol. Este valor evita que los árboles sean demasiado complejos y que el modelo se ajuste excesivamente al ruido de los datos.
- `subsample = 0.8`: porcentaje de datos de entrenamiento que se usa para construir cada árbol, fomentando la aleatoriedad y la generalización.
- `colsample_bytree = 0.8`: porcentaje de variables que se utiliza para cada árbol, lo que promueve la diversidad de características y reduce el riesgo de dependencia de un subconjunto pequeño de variables.
- `random_state = 42`: garantiza la reproducibilidad de los resultados.
- `eval_metric = 'logloss'`: métrica utilizada para evaluar el modelo durante el entrenamiento; es adecuada para problemas de clasificación binaria.

Modelo	Accuracy	F1 weighted	ROC AUC
Random Forest	0.9208	0.9186	0.9422
Regresión lógistica	0.9010	0.8995	0.9463
XGBoost	0.9257	0.9239	0.9195

Con los resultados obtenidos, se puede indicar que el modelo XGBoost presentó el mejor desempeño general, alcanzando una mayor Accuracy (0.9257) y F1-score (0.9239).

Aunque su ROC AUC (0.9195) fue ligeramente inferior al de la Regresión Logística (0.9463), esto indica que XGBoost acierta con mayor frecuencia en la clasificación general, mientras que la regresión Logística tiene una ligeramente mejor capacidad para distinguir entre clases .

El modelo Random Forest también mostró un rendimiento sólido, con buena precisión y equilibrio entre clases.

Por su parte, la Regresión Logística resulta una opción útil cuando se busca interpretabilidad y simplicidad, aun cuando su desempeño en Accuracy y F1-score es ligeramente menor.

Evidencias del proyecto

- ML canvas
https://github.com/secabezon/MLOps/blob/main/docs/MLCanvas_South_German_Credit.docx
- Github <https://github.com/secabezon/MLOps>
- Dagshub <https://dagshub.com/Pamela-ruiz9/MLOps>
- EDA notebook
https://github.com/secabezon/MLOps/blob/main/notebooks/EDA_german_credit.ipynb
- Modeling notebook
https://github.com/secabezon/MLOps/blob/main/notebooks/modeling_and_training.ipynb
- Presentación https://tecmx-my.sharepoint.com/:p/r/personal/a01840261_tec_mx/_layouts/15/Doc.aspx?siteId=7BD0B2759E-10D4-471D-B482-92EED3F483C4%7D&file=Presentaci%25u00f3n.pptx&action=edit&mobileredirect=true
- Video <https://youtu.be/8-Dujrs5PNo>

Conclusiones

Este proyecto implementó un flujo completo de Machine Learning y MLOps, abarcando desde la limpieza y preparación de datos hasta el entrenamiento y versionado de modelos. Se garantizó la reproducibilidad mediante el uso de DVC y Dagshub para el manejo de datos y artefactos, y Git para el control de código. El proceso de limpieza incluyó múltiples etapas de normalización, imputación, detección de outliers y aseguramiento de tipos de datos coherentes. Asimismo, se entrenaron diversos modelos de clasificación que proporcionaron una base sólida para predecir el riesgo crediticio, evaluados con métricas interpretables y comparables como ROC-AUC y F1-Score. En conjunto, el proyecto demuestra una aplicación práctica de técnicas de Machine Learning explicables, estructuradas y trazables, con potencial real en entornos financieros.

Próximos Pasos (Fase 2)

Próximos pasos Fase 2:

- Formalizar pipeline con dvc.yaml (stages prep/train) y params.yaml.
- Registrar/comparar experimentos con MLflow.
- Exponer el modelo con FastAPI (contrato de entrada/salida).
- Contenerizar con Docker y preparar despliegue.
- Monitoreo inicial: plan para data drift / performance.

Referencias (APA 7^a edición)

Dua, D., & Graff, C. (2019). *UCI Machine Learning Repository: Statlog (German Credit Data)*. University of California, Irvine. Recuperado de
[https://archive.ics.uci.edu/ml/datasets/statlog+\(german+credit+data\)](https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data))

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

<https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>

DVC. (2023). *Data Version Control Documentation*. Iterative.ai. Recuperado de
<https://dvc.org>

DagsHub. (2023). *Collaborative data science and versioning platform*. Recuperado de
<https://dagshub.com>

MLflow. (2023). *MLflow: An open source platform for the machine learning lifecycle*. Recuperado de <https://mlflow.org>