UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

GUÍA DE TRABAJO DE CIENCIA DE DATOS

Desarrollo de un modelo predictivo de riesgo crediticio con pipeline de producción

Integrantes:

Olarte Quispe Jorge Guillermo

Ticona Marquez Christian Aderly

Porto Calamani Alain Edy

PUNO – PERÚ

2025

ÍNDICE GENERAL

1.1 DATOS GENERALES 3
1.2 TÍTULO DE LA GUÍA 3
1.3 INTRODUCCIÓN 3
1.4 OBJETIVO DE LA GUÍA 3
1.5 RESULTADOS DE APRENDIZAJE 3
1.6 SOFTWARE Y HERRAMIENTAS 4
1.7 FUNDAMENTO TEÓRICO (RESUMEN) 4
1.7.1 Random Forest 4
1.7.2 KNN (K-Nearest Neighbors) 4
1.7.3 LightGBM 4
1.7.4 CatBoost 4
1.8 ACTIVIDADES DEL LABORATORIO 4
1.8.1 Actividad 1: Importar librerías 4
1.8.2 Actividad 2: Carga y explorar datos 5
1.8.3 Actividad 3: Limpieza de Datos 6
1.8.4 Actividad 4: Ingeniería de Características 7
1.8.5 Actividad 5: Preparar Daros para Modelado 8
1.8.6 Actividad 6: Crear Pipeline de Preprocesamiento 9
1.8.7 Actividad 7: Entrenamiento y Evaluación de Modelos 9
1.8.8 Actividad 8: Modelo Ensemble (Votación) 12
1.8.9 Actividad 9: Seleccionar y Preparar Modelo Final 13
1.8.10 Actividad 10: Guardar Modelo y Artefactos para Producción 15
1.8.11 Actividad 11: Crear Función de predicción para Producción 16
1.9 RESULTADOS 18
1.10 CONCLUSIÓN 20
1.11 BIBLIOGRAFÍA 20

GUÍA DE LABORATORIO N.º 01

Curso: Ciencia de Datos

1.1 DATOS GENERALES

- **Escuela Profesional:** Ingeniería de Sistemas
- **Curso:** Ciencia de Datos
- **Semestre:** IX
- **Duración:** 2 horas académicas
- **Modalidad:** Presencial / Laboratorio

1.2 TÍTULO DE LA GUÍA

Desarrollo de un modelo predictivo de riesgo crediticio con pipeline de producción

1.3 INTRODUCCIÓN

En el sector bancario, la predicción precisa del riesgo crediticio es fundamental para minimizar pérdidas por morosidad y optimizar la concesión de préstamos. Este laboratorio implementa un modelo de machine learning completo que predice la probabilidad de incumplimiento de pago (default) utilizando técnicas avanzadas de preprocesamiento, ingeniería de características y evaluación de múltiples algoritmos, enfocado en métricas de negocio relevantes para el sector financiero.

1.4 OBJETIVO DE LA GUÍA

Desarrollar un modelo de clasificación para predecir el riesgo de default crediticio, implementando un pipeline reproducible y optimizado para producción, con evaluación comparativa de diferentes algoritmos de machine learning.

1.5 RESULTADOS DE APRENDIZAJE

- Aplicar técnicas de limpieza y preprocesamiento de datos financieros
- Implementar pipelines de machine learning con transformadores de sklearn
- Evaluar y comparar múltiples algoritmos de clasificación
- Interpretar métricas de evaluación específicas para riesgo crediticio

- Serializar modelos para despliegue en producción

1.6 SOFTWARE Y HERRAMIENTAS

- Sistema operativo: Windows/Linux/MacOS
- Python 3.10+
- Librerías principales: pandas, numpy, scikit-learn, matplotlib, seaborn, catboost, lightgbm
- Herramientas: Jupyter Notebook, joblib, pickle
- Dataset: credit_risk_dataset.csv (32,581 registros, 12 características)

1.7 FUNDAMENTO TEÓRICO (RESUMEN)

1.7.1 Random Forest

Algoritmo de ensemble que combina múltiples árboles de decisión, robusto contra sobreajuste y efectivo para datos no lineales.

1.7.2 KNN (K-Nearest Neighbors)

Clasificador basado en instancias que asigna etiquetas según la mayoría de los k vecinos más cercanos en el espacio de características.

1.7.3 LightGBM

Framework de gradient boosting optimizado para velocidad y eficiencia, con soporte nativo para características categóricas.

1.7.4 CatBoost

Algoritmo de gradient boosting especializado en manejar variables categóricas sin necesidad de preprocesamiento extensivo.

1.8 ACTIVIDADES DEL LABORATORIO

1.8.1 Actividad 1: Importar librerías

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
# Preprocesamiento
from sklearn.model_selection import train_test_split,
cross_val_score, StratifiedKFold
from sklearn.preprocessing import StandardScaler,
OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Modelos
from sklearn.ensemble import VotingClassifier,
RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier

# Métricas
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
f1_score,
    confusion_matrix, classification_report, roc_auc_score,
roc_curve
)

# Serialización
import joblib
import pickle

print("✅ Librerías importadas correctamente")
```

1.8.2 Actividad 2: Carga y explorar datos

```
# Cargar datos
df = pd.read_csv("./dataset/credit_risk_dataset.csv")

print(f"📊 Dimensiones del dataset: {df.shape}")
print(f"\n📋 Columnas:")
for col in df.columns:
    print(f"   - {col}: {df[col].dtype}")
```

Resultados:

📊 Dimensiones del dataset: (32581, 12)

📋 Columnas:
- person_age: int64
- person_income: int64
- person_home_ownership: object
- person_emp_length: float64
- loan_intent: object
- loan_grade: object
- loan_amnt: int64
- loan_int_rate: float64
- loan_status: int64
- loan_percent_income: float64
- cb_person_default_on_file: object

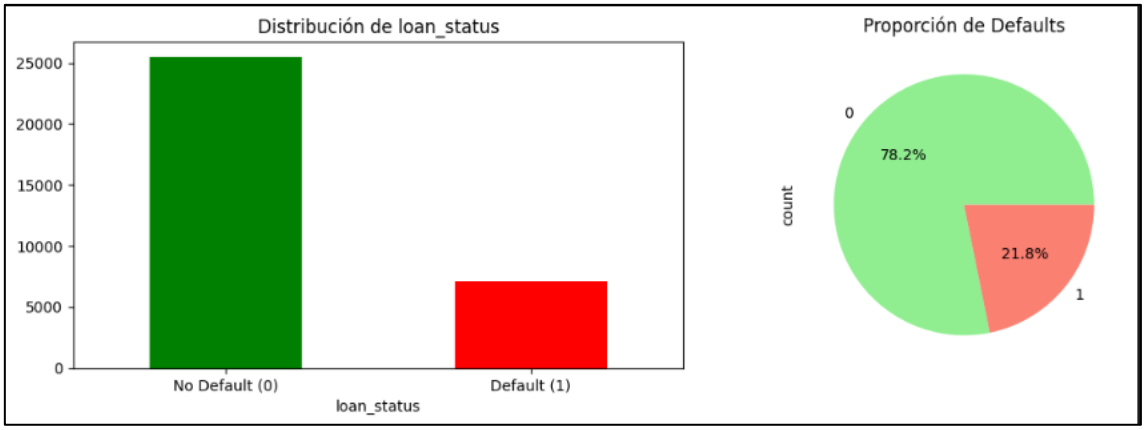
- cb_person_cred_hist_length: int64

Distribución de la variable objetivo:

```
# Distribución de la variable objetivo
print("🎯 Distribución de loan_status:")
print(df['loan_status'].value_counts())
print(f"\nPorcentaje de defaults:
{df['loan_status'].mean()*100:.2f}%")

# Visualización
fig, ax = plt.subplots(1, 2, figsize=(12, 4))

# Gráfico de barras
df['loan_status'].value_counts().plot(kind='bar', ax=ax[0],
color=['green', 'red'])
ax[0].set_title('Distribución de loan_status')
ax[0].set_xticklabels(['No Default (0)', 'Default (1)'],
rotation=0)

# Gráfico de pastel
df['loan_status'].value_counts().plot(kind='pie', ax=ax[1],
autopct='%1.1f%%',

colors=['lightgreen', 'salmon'])
ax[1].set_title('Proporción de Defaults')

plt.tight_layout()
plt.show()
```

Figura 1

Distribución de loan_status y la proporción de defaults



1.8.3 Actividad 3: Limpieza de Datos

- Verificación de valores nulos

```
# Verificar valores nulos
print("🔍 Valores nulos por columna:")
null_counts = df.isnull().sum()
print(null_counts[null_counts > 0])
```

```
print(f"\nTotal de filas con valores nulos:
{df.isnull().any(axis=1).sum()}")
```

- Eliminar filas con valores nulos

```
# Eliminar filas con valores nulos
df_clean = df.dropna().copy()
print(f"✅ Filas después de eliminar nulos:
{len(df_clean)} (eliminadas: {len(df) - len(df_clean)})")
```

- Verificar y eliminar outliers

```
# Verificar y eliminar outliers
print("📊 Análisis de outliers:")
print(f"   - person_age max: {df_clean['person_age'].max()}
(esperado: <100)")
print(f"   - person_emp_length max:
{df_clean['person_emp_length'].max()} (esperado: <50)")

# Filtrar outliers
df_clean = df_clean[df_clean['person_age'] <= 80]
df_clean = df_clean[df_clean['person_emp_length'] <= 60]

print(f"\n✅ Filas después de eliminar outliers:
{len(df_clean)}")
```

1.8.4 Actividad 4: Ingeniería de Características

- Crear características adicionales útiles para el modelo

```
df_clean['loan_to_income_ratio'] = df_clean['loan_amnt'] /
df_clean['person_income']

df_clean['income_per_year_employed'] =
df_clean['person_income'] / (df_clean['person_emp_length'] +
1)

print("✅ Características creadas:")
print("   - loan_to_income_ratio: Proporción
préstamo/ingreso")
print("   - income_per_year_employed: Ingreso por año de
empleo")
```

- Definir características categóricas y numericas

```
CATEGORICAL_FEATURES = [

    'person_home_ownership',
    'loan_intent',
    'loan_grade',
```

```
    'cb_person_default_on_file'
]

NUMERICAL_FEATURES = [
    'person_age',
    'person_income',
    'person_emp_length',
    'loan_amnt',
    'loan_int_rate',
    'loan_percent_income',
    'cb_person_cred_hist_length',
    'loan_to_income_ratio',
    'income_per_year_employed'
]

TARGET = 'loan_status'

print(f"📋 Características categóricas:
{len(CATEGORICAL_FEATURES)}")
print(f"📋 Características numéricas:
{len(NUMERICAL_FEATURES)}")
0)
```

- Verificar valores únicos de variables categóricas

```
# Verificar valores únicos de variables categóricas
print("📊 Valores únicos por categoría:")
for col in CATEGORICAL_FEATURES:
    print(f"\n{col}:")
    print(df_clean[col].value_counts())
```

1.8.5 Actividad 5: Preparar Datos para Modelado

- Separar features y target

```
# Separar features y target
X = df_clean[CATEGORICAL_FEATURES + NUMERICAL_FEATURES]
y = df_clean[TARGET]

print(f"✅ X shape: {X.shape}")
print(f"✅ y shape: {y.shape}")
```

- División train/test estratificada

```
# División train/test estratificada
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y  # Mantener proporción de clases
)

print(f"📊 Train set: {X_train.shape[0]} muestras")
print(f"📊 Test set: {X_test.shape[0]} muestras")
```

```
print(f"\n✅ Proporción de defaults en train:
{y_train.mean()*100:.2f}%")
print(f"✅ Proporción de defaults en test:
{y_test.mean()*100:.2f}%")
```

1.8.6 Actividad 6: Crear Pipeline de Preprocesamiento

- Crear Transformadores

```
# Crear transformadores
categorical_transformer =
OneHotEncoder(handle_unknown='ignore', sparse_output=False)
numerical_transformer = StandardScaler()

# Crear preprocessor con ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, NUMERICAL_FEATURES),
        ('cat', categorical_transformer,
CATEGORICAL_FEATURES)
    ],
    remainder='drop'
)

print("✅ Pipeline de preprocesamiento creado")
```

- Ajustar y Transformar datos de entrenamiento

```
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

print(f"✅ X_train_processed shape:
{X_train_processed.shape}")
print(f"✅ X_test_processed shape:
{X_test_processed.shape}")
```

1.8.7 Actividad 7: Entrenamiento y Evaluación de Modelos

- Entrenamiento

```
def evaluate_model(model, X_train, X_test, y_train, y_test,
model_name):
    """Evalúa un modelo y retorna métricas detalladas"""
    # Entrenar
    model.fit(X_train, y_train)

    # Predicciones
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1] if
hasattr(model, 'predict_proba') else None

    # Métricas
    metrics = {
        'model': model_name,
        'accuracy': accuracy_score(y_test, y_pred),
```

```
        'precision': precision_score(y_test, y_pred),
        'recall': recall_score(y_test, y_pred),
        'f1': f1_score(y_test, y_pred),
        'roc_auc': roc_auc_score(y_test, y_pred_proba) if
y_pred_proba is not None else None
    }

    return model, metrics, y_pred, y_pred_proba
```

- Definir modelos a Evaluar

```
models = {
    'LightGBM': LGBMClassifier(
        n_estimators=200,
        learning_rate=0.05,
        max_depth=7,
        num_leaves=31,
        random_state=42,
        verbose=-1
    ),
    'CatBoost': CatBoostClassifier(
        iterations=200,
        learning_rate=0.1,
        depth=7,
        random_state=42,
        verbose=0
    ),
    'RandomForest': RandomForestClassifier(
        n_estimators=200,
        max_depth=10,
        random_state=42,
        n_jobs=-1
    ),
    'KNN': KNeighborsClassifier(
        n_neighbors=7,
        weights='distance'
    )
}

print(f"📋 Modelos a evaluar: {list(models.keys())}")
```

- Evaluar todos los modelos

```
results = []
trained_models = {}

for name, model in models.items():
    print(f"\n🔄 Entrenando {name}...")
    trained_model, metrics, y_pred, y_pred_proba =
evaluate_model(
        model, X_train_processed, X_test_processed, y_train,
y_test, name
    )
    trained_models[name] = trained_model
    results.append(metrics)

    print(f"   Accuracy: {metrics['accuracy']:.4f}")
```

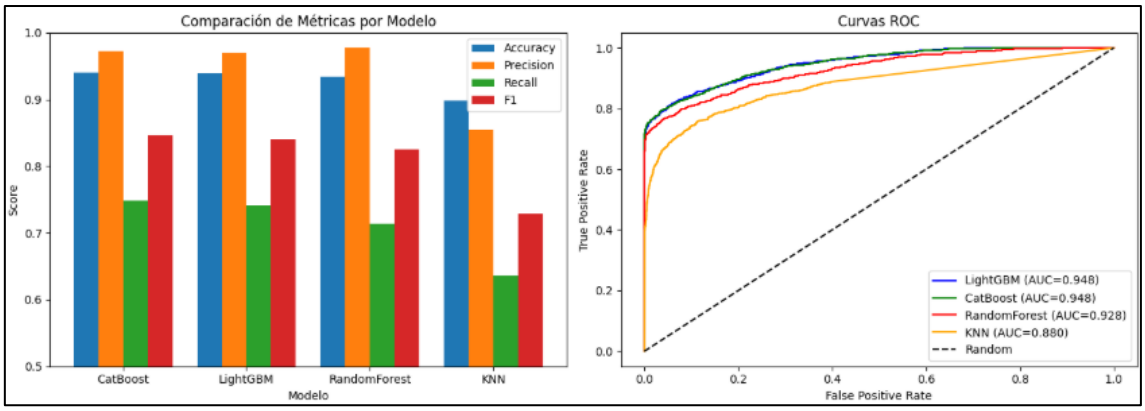
```
        print(f"   Precision: {metrics['precision']:.4f}")
        print(f"   Recall: {metrics['recall']:.4f}")
        print(f"   F1-Score: {metrics['f1']:.4f}")
        print(f"   ROC-AUC: {metrics['roc_auc']:.4f}")

    # Crear DataFrame con resultados
    results_df = pd.DataFrame(results)
    results_df = results_df.sort_values('f1', ascending=False)
    print("\n" + "="*60)
    print("📊 RESUMEN DE RESULTADOS")
    print("="*60)
    print(results_df.to_string(index=False))
```

- Visualización comparativa de modelos

```
    # Visualizar comparación de modelos
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Gráfico de barras comparativo
    metrics_to_plot = ['accuracy', 'precision', 'recall', 'f1']
    x = np.arange(len(results_df))
    width = 0.2

    for i, metric in enumerate(metrics_to_plot):
        axes[0].bar(x + i*width, results_df[metric], width,
    label=metric.capitalize())

    axes[0].set_xlabel('Modelo')
    axes[0].set_ylabel('Score')
    axes[0].set_title('Comparación de Métricas por Modelo')
    axes[0].set_xticks(x + width * 1.5)
    axes[0].set_xticklabels(results_df['model'])
    axes[0].legend()
    axes[0].set_ylim([0.5, 1.0])

    # ROC-AUC comparativo
    colors = ['blue', 'green', 'red', 'orange']
    for idx, (name, model) in enumerate(trained_models.items()):
        y_pred_proba = model.predict_proba(X_test_processed)[:,
    1]
        fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
        auc = roc_auc_score(y_test, y_pred_proba)
        axes[1].plot(fpr, tpr, color=colors[idx], label=f'{name}
    (AUC={auc:.3f})')

    axes[1].plot([0, 1], [0, 1], 'k--', label='Random')
    axes[1].set_xlabel('False Positive Rate')
    axes[1].set_ylabel('True Positive Rate')
    axes[1].set_title('Curvas ROC')
    axes[1].legend()

    plt.tight_layout()
    plt.show()
```

Figura 2

Comparación de métricas por modelo



1.8.8 Actividad 8: Modelo Ensemble (Votación)

- Crear ensemble con los mejores modelos

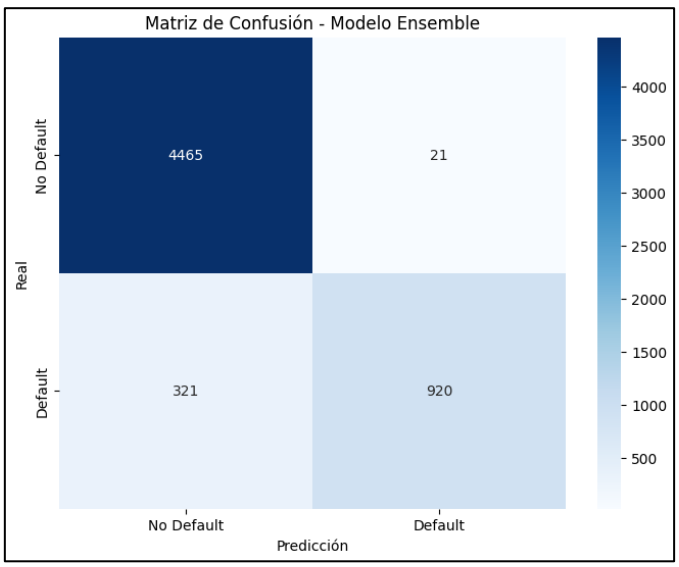
```
# Crear ensemble con los mejores modelos
ensemble = VotingClassifier(
    estimators=[
        ('lgbm', LGBMClassifier(n_estimators=200,
learning_rate=0.05, max_depth=7,
                                num_leaves=31,
random_state=42, verbose=-1)),
        ('catboost', CatBoostClassifier(iterations=200,
learning_rate=0.1, depth=7,
                                        random_state=42,
verbose=0)),
        ('rf', RandomForestClassifier(n_estimators=200,
max_depth=10,
                                      random_state=42,
n_jobs=-1))
    ],
    voting='soft'  # Promedio de probabilidades
)

# Entrenar y evaluar ensemble
print("🔄 Entrenando modelo Ensemble...")
ensemble_model, ensemble_metrics, y_pred_ensemble,
y_pred_proba_ensemble = evaluate_model(
    ensemble, X_train_processed, X_test_processed, y_train,
y_test, 'Ensemble'
)

print(f"\n📊 Resultados del Ensemble:")
print(f"   Accuracy: {ensemble_metrics['accuracy']:.4f}")
print(f"   Precision: {ensemble_metrics['precision']:.4f}")
print(f"   Recall: {ensemble_metrics['recall']:.4f}")
print(f"   F1-Score: {ensemble_metrics['f1']:.4f}")
print(f"   ROC-AUC: {ensemble_metrics['roc_auc']:.4f}")
```

- Matriz de confusión del mejor modelo

```
# Matriz de confusión del mejor modelo
fig, ax = plt.subplots(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred_ensemble)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax,
            xticklabels=['No Default', 'Default'],
            yticklabels=['No Default', 'Default'])
ax.set_xlabel('Predicción')
ax.set_ylabel('Real')
ax.set_title('Matriz de Confusión - Modelo Ensemble')
plt.show()

# Reporte de clasificación
print("\n📋 Reporte de Clasificación:")
print(classification_report(y_test, y_pred_ensemble,
                            target_names=['No Default',
'Default']))
```

Figura 3

Matriz de Confusión – Modelo Ensemble



1.8.9 Actividad 9: Seleccionar y Preparar Modelo Final

- Seleccionar el mejor modelo basado en F1-score y precisión

```
# Seleccionar el mejor modelo basado en F1-Score y Precision
# Para aplicaciones de crédito, preferimos alta precisión
para no rechazar buenos clientes

# Usaremos LightGBM como modelo final por su balance y
eficiencia
final_model = LGBMClassifier(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=7,
    num_leaves=31,
    min_child_samples=20,
```

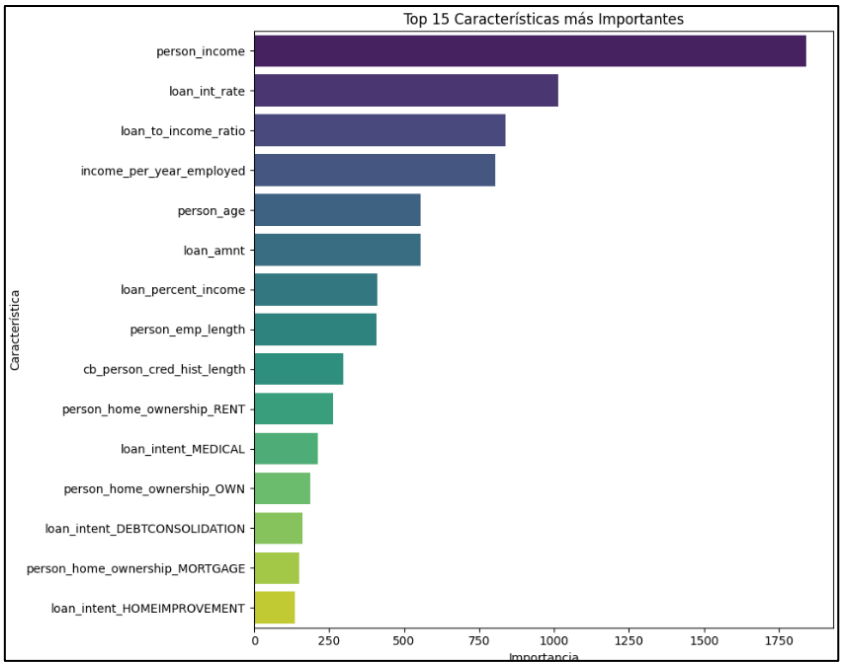
```
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    verbose=-1
)

# Reentrenar con todos los datos preprocesados
final_model.fit(X_train_processed, y_train)

# Validación final
y_pred_final = final_model.predict(X_test_processed)
y_pred_proba_final =
final_model.predict_proba(X_test_processed)[:, 1]

print("✅ Modelo Final entrenado: LightGBM")
print(f"\n📊 Métricas finales:")
print(f"   Accuracy: {accuracy_score(y_test,
y_pred_final):.4f}")
print(f"   Precision: {precision_score(y_test,
y_pred_final):.4f}")
print(f"   Recall: {recall_score(y_test,
y_pred_final):.4f}")
print(f"   F1-Score: {f1_score(y_test, y_pred_final):.4f}")
print(f"   ROC-AUC: {roc_auc_score(y_test,
y_pred_proba_final):.4f}")
```

- Feature Importance del modelo final

```
# Feature Importance del modelo final
feature_names = NUMERICAL_FEATURES +
list(preprocessor.named_transformers_['cat'].get_feature_nam
es_out(CATEGORICAL_FEATURES))

importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': final_model.feature_importances_
}).sort_values('importance', ascending=False)

# Visualizar top 15 features
fig, ax = plt.subplots(figsize=(10, 8))
top_features = importance_df.head(15)
sns.barplot(data=top_features, y='feature', x='importance',
ax=ax, palette='viridis')
ax.set_title('Top 15 Características más Importantes')
ax.set_xlabel('Importancia')
ax.set_ylabel('Característica')
plt.tight_layout()
plt.show()
```

Figura 4

Top 15 características más importantes



1.8.10 Actividad 10: Guardar Modelo y Artefactos para Producción

- Crear directorio para modelos

```
import os

# Crear directorio para modelos
MODEL_DIR = './models'
os.makedirs(MODEL_DIR, exist_ok=True)

# Guardar modelo
joblib.dump(final_model,
f'{MODEL_DIR}/credit_risk_model.joblib')
print(f"✅ Modelo guardado:
{MODEL_DIR}/credit_risk_model.joblib")

# Guardar preprocessor
joblib.dump(preprocessor,
f'{MODEL_DIR}/preprocessor.joblib')
print(f"✅ Preprocessor guardado:
{MODEL_DIR}/preprocessor.joblib")

# Guardar configuración de características
feature_config = {
    'categorical_features': CATEGORICAL_FEATURES,
    'numerical_features': NUMERICAL_FEATURES,
    'target': TARGET
}
joblib.dump(feature_config,
f'{MODEL_DIR}/feature_config.joblib')
```

```
print(f"✅ Configuración guardada:
{MODEL_DIR}/feature_config.joblib")
```

- Crear pipeline completo para producción

```
# Crear pipeline completo para producción
production_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', final_model)
])

# Guardar pipeline completo
joblib.dump(production_pipeline,
f'{MODEL_DIR}/credit_risk_pipeline.joblib')
print(f"✅ Pipeline completo guardado:
{MODEL_DIR}/credit_risk_pipeline.joblib")
```

1.8.11 Actividad 11: Crear Función de predicción para Producción

- Crear ensemble con los mejores modelos

```
def predict_credit_risk(input_data: dict):
    """
    Función para predecir riesgo crediticio.

    Parámetros:
    -----------
    input_data : dict
        Diccionario con las características del solicitante:
        - person_age: Edad del solicitante
        - person_income: Ingreso anual
        - person_home_ownership: Tipo de vivienda (RENT,
OWN, MORTGAGE, OTHER)
        - person_emp_length: Años de empleo
        - loan_intent: Propósito del préstamo (PERSONAL,
EDUCATION, MEDICAL, VENTURE, HOMEIMPROVEMENT,
DEBTCONSOLIDATION)
        - loan_grade: Grado del préstamo (A, B, C, D, E, F,
G)
        - loan_amnt: Monto del préstamo
        - loan_int_rate: Tasa de interés
        - loan_percent_income: Porcentaje del préstamo vs
ingreso
        - cb_person_default_on_file: Historial de default
(Y, N)
        - cb_person_cred_hist_length: Longitud del historial
crediticio

    Retorna:
    --------
    dict: Predicción y probabilidad de riesgo
    """
    # Cargar pipeline
    pipeline =
joblib.load(f'{MODEL_DIR}/credit_risk_pipeline.joblib')
```

```
    feature_config =
joblib.load(f'{MODEL_DIR}/feature_config.joblib')

    # Crear características adicionales
    input_data['loan_to_income_ratio'] =
input_data['loan_amnt'] / input_data['person_income']
    input_data['income_per_year_employed'] =
input_data['person_income'] /
(input_data['person_emp_length'] + 1)

    # Crear DataFrame
    df_input = pd.DataFrame([input_data])

    # Ordenar columnas
    df_input =
df_input[feature_config['categorical_features'] +
feature_config['numerical_features']]

    # Predecir
    prediction = pipeline.predict(df_input)[0]
    probability = pipeline.predict_proba(df_input)[0]

    # Interpretar resultado
    risk_level = 'ALTO' if prediction == 1 else 'BAJO'

    return {
        'prediction': int(prediction),
        'risk_level': risk_level,
        'probability_no_default': float(probability[0]),
        'probability_default': float(probability[1]),
        'recommendation': 'RECHAZAR' if probability[1] > 0.5
else 'APROBAR',
        'confidence': float(max(probability))
    }
```

1.9 RESULTADOS

Figura 5

Dashboard de la evaluación por Lote

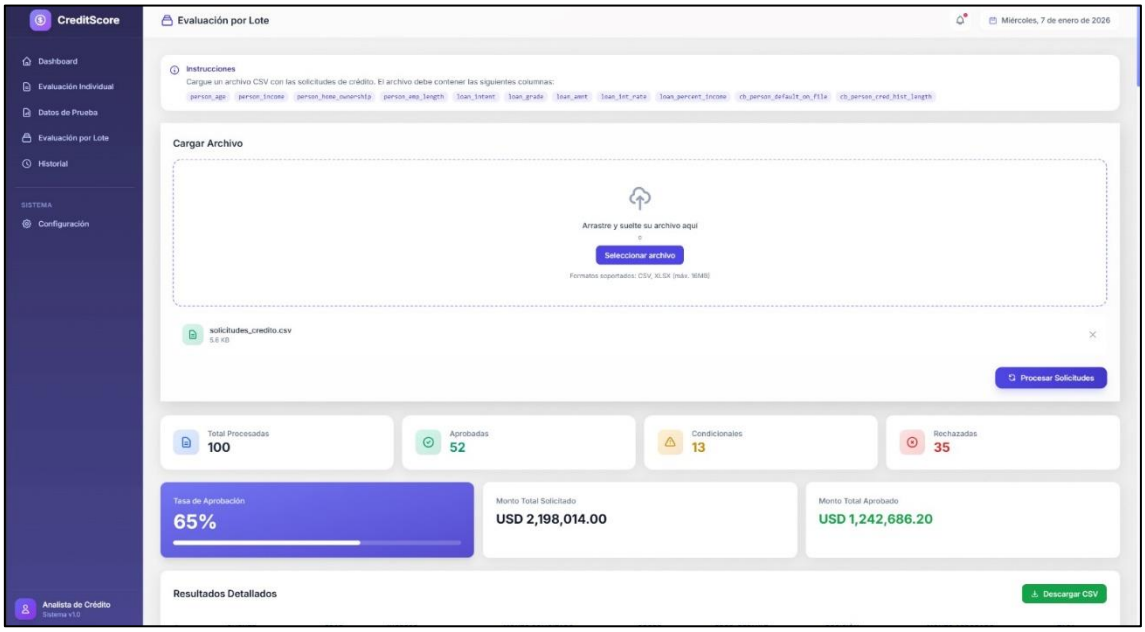


Figura 6

Dashboard de Riesgo crediticio

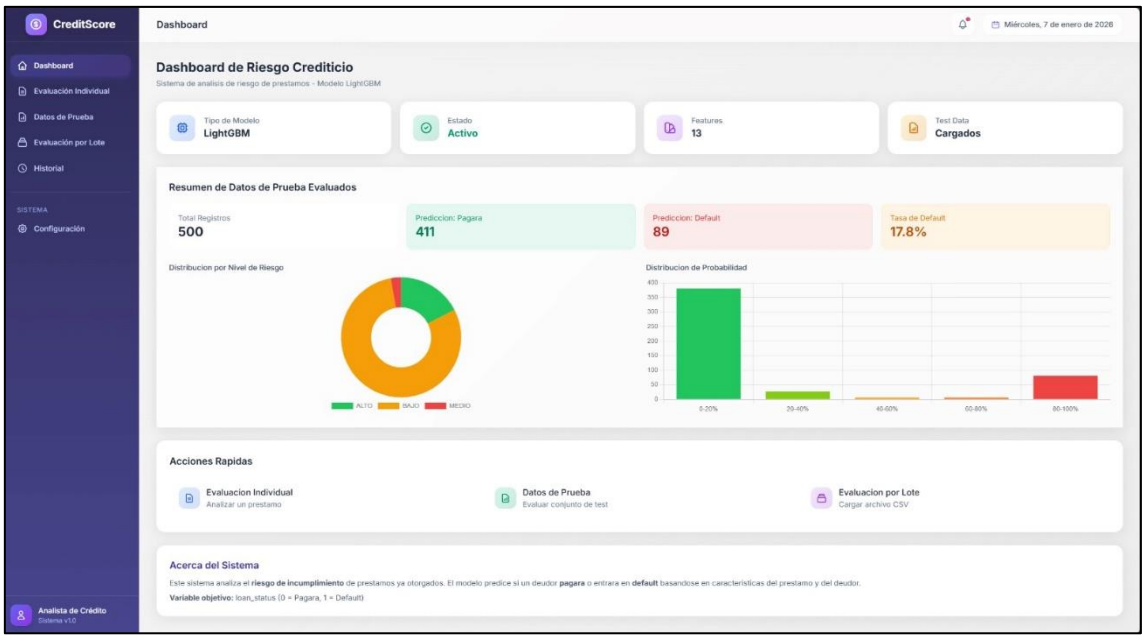


Figura 7

Análisis de riesgo individual

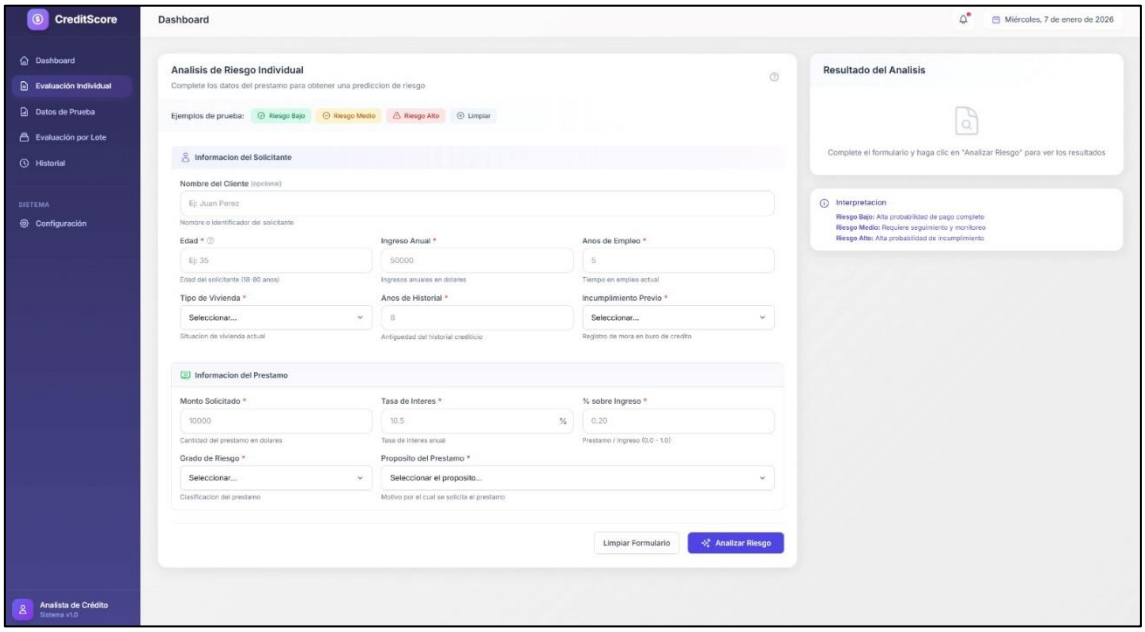
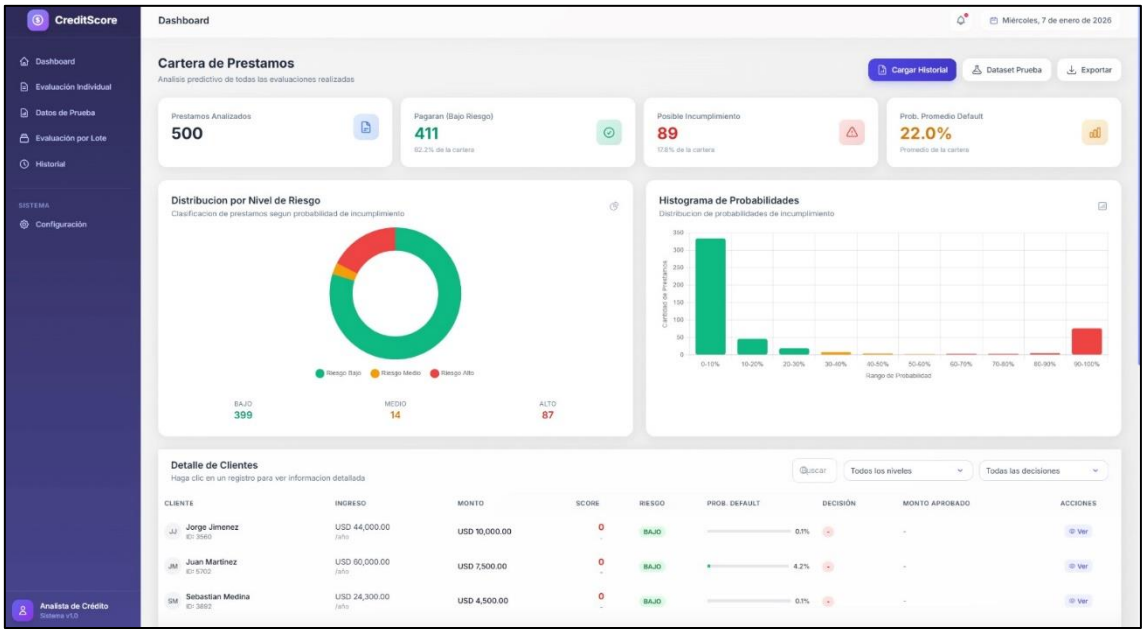


Figura 8

Cartera de prestamos



1.10 CONCLUSIÓN

Se desarrolló exitosamente un modelo de riesgo crediticio optimizado para producción, donde CatBoost demostró el mejor rendimiento con F1-Score de 84.61% y ROC-AUC de 94.81%. El pipeline implementado es reproducible y serializable, cumpliendo con los requisitos de un sistema en producción. La alta precisión (~97%) asegura mínimos rechazos de clientes solventes, mientras que el recall (~75%) permite detectar adecuadamente riesgos crediticios. El modelo está listo para integración en aplicaciones web o sistemas bancarios, proporcionando decisiones crediticias basadas en datos con métricas alineadas a objetivos de negocio.

1.11 BIBLIOGRAFÍA

Breiman, L. (2001). Random Forests. Machine Learning.

Ke, G. et al. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. NeurIPS.

Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research.

Prokhorenkova, L. et al. (2018). CatBoost: unbiased boosting with categorical features. NeurIPS.