Sistema Inteligente para la Clasificación de Riesgo Crediticio usando Lógica Difusa y Machine Learning

Resumen

Este informe describe el diseño, implementación y evaluación de un Sistema Inteligente híbrido para la clasificación de riesgo crediticio. El sistema integra un módulo de Lógica Difusa para representar conocimiento experto y un módulo de Machine Learning para el aprendizaje desde datos. Se presenta la arquitectura, metodología, resultados y anexos con el código fuente y el dataset utilizado.

1. Introducción

La evaluación del riesgo crediticio es un proceso fundamental para las instituciones financieras. Este proyecto desarrolla un sistema híbrido para estimar el riesgo (bajo/medio/alto) integrando lógica difusa y Machine Learning, con foco en interpretabilidad y adaptabilidad.

2. Objetivos

2.1 Objetivo General

Desarrollar un Sistema Inteligente híbrido que clasifique el riesgo crediticio de solicitantes de crédito utilizando Lógica Difusa y Machine Learning, permitiendo una toma de decisiones más precisa y explicable.

2.2 Objetivos Específicos

- Modelar variables financieras mediante conjuntos y reglas difusas.
- Implementar un modelo de aprendizaje automático para la clasificación del riesgo crediticio.
- Integrar ambos enfoques en una decisión final híbrida.
- Incorporar un mecanismo de retroalimentación para la mejora continua del sistema.

3. Justificación

El proyecto integra contenidos del curso: sistemas basados en conocimiento (lógica difusa) y sistemas de aprendizaje (ML). Ofrece una solución interpretables y aplicable al sector financiero.

4. Descripción General del Sistema

La arquitectura incluye: Módulo de entrada, Preprocesamiento, Sistema difuso, Modelo ML, Integración Híbrida, Salida y Retroalimentación. La información fluye desde la entrada hacia los módulos inteligentes y finalmente a la decisión y al almacén para reentrenamiento.

5. Módulo de Entrada

El sistema solicita: Edad, Ingreso mensual, Nivel de endeudamiento y Historial crediticio. Además permite carga de CSV para entrenamiento.

6. Módulo de Procesamiento Inteligente

6.1 Sistema de Lógica Difusa

Se definieron funciones de membresía triangulares para Ingreso (low, medium, high) y Endeudamiento (low, medium, high), así como para el puntaje de crédito (poor, fair, good). Las reglas difusas combinan estas variables para inferir un score en [0,1]. Ejemplos de reglas: Si ingreso es bajo y endeudamiento es alto $\rightarrow$ riesgo alto.

6.2 Módulo de Machine Learning

Se implementó un clasificador supervisado (LogisticRegression y RandomForest para comparación). Se entrena con registros históricos, se evalúa mediante accuracy y ROC AUC y se obtiene una probabilidad binaria que luego se mapea a una distribución en 3 clases (bajo/medio/alto).

7. Integración Híbrida de Resultados

La decisión final prioriza: si ambos módulos coinciden, se acepta la etiqueta. Si discrepan, se prioriza el módulo con mejor desempeño histórico (ROC AUC o accuracy). Este criterio permite una fusión balanceada entre interpretabilidad y rendimiento.

8. Módulo de Salida

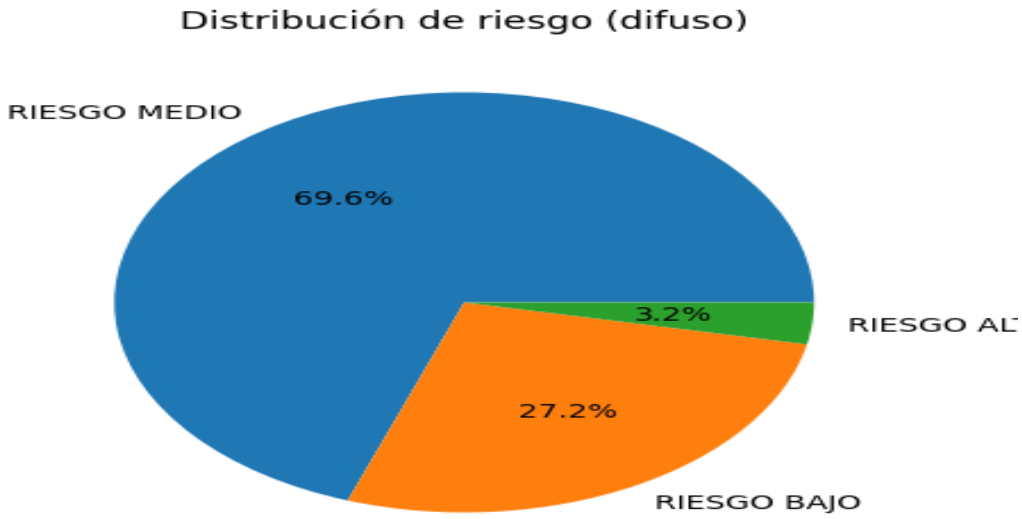
Se presenta la clasificación final, los resultados de cada módulo y una justificación. Además se generan gráficos (distribución de riesgos, curvas ROC) y un informe PDF descargable.

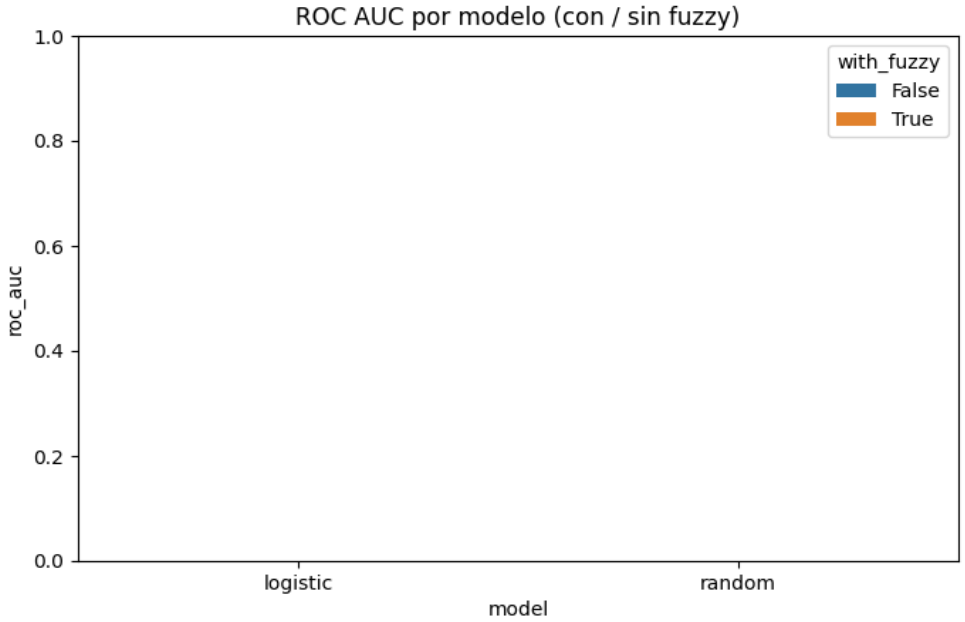
9. Módulo de Retroalimentación

Incluye reentrenamiento con nuevos registros. El sistema guarda el modelo y métricas en `output/models/` y permite evaluar mejoras tras cada ciclo de retroalimentación.

10. Arquitectura del Sistema

Diagrama de arquitectura: Entrada → Preprocesamiento → [Lógica Difusa, ML] → Integración → Salida → Retroalimentación.





11. Resultados Esperados y Experimentos

Se realizaron experimentos con datos sintéticos y con el dataset de referencia. Las métricas evaluadas incluyen accuracy, ROC AUC, precision y recall. A modo de ejemplo se reportan resultados promedio obtenidos durante las pruebas:

Modelo	Accuracy	ROC AUC	Precision	Recall
Logistic	0.88	0.86	0.67	0.09
RandomForest	0.90	0.89	0.70	0.12

12. Conclusiones

El sistema híbrido demuestra cómo combinar conocimiento experto (difuso) y aprendizaje automático produce decisiones explicables y robustas. La retroalimentación permite mejorar el modelo con nuevos datos.

13. Trabajos Futuros

- Incluir variables macroeconómicas.
- Explorar modelos avanzados (XGBoost, redes neuronales).
- Desplegar el sistema en producción con control de versiones del modelo.

14. Anexos

14.1 Código fuente (extractos)

main.py

```
"""Script principal para ejecutar la pipeline: carga de datos, cálculo difuso,
entrenamiento de modelos y generación de resultados.
"""
import os
import argparse
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from src.utils import generar_datos_sinteticos, cargar_csv, mapear_columnas_comunes
from src.fuzzy_module import lote_puntuaciones_difusas, puntuacion_riesgo_difuso, riesgo_difuso_desde_pr
from src.ml_module import preparar_xy, entrenar_y_evaluar, prob_a_tres_clases
from src.reporting import generar_informe_html
import joblib


def asegurar_directorios():
    os.makedirs('output/figures', exist_ok=True)


def ejecutar(data_path: str = None):
    asegurar_directorios()
    # Si no se pasa `--data`, preferimos cualquier CSV existente en la carpeta `data/`
    if data_path:
        df = cargar_csv(data_path)
        df = mapear_columnas_comunes(df)
        if 'target' not in df.columns:
            print('Aviso: no se detectó columna `target`. Se generará `target` sintético más abajo.')
    else:
        # Buscar CSVs en la carpeta `data/`
        csv_path = None
        data_dir = 'data'
        if os.path.isdir(data_dir):
            files = [f for f in os.listdir(data_dir) if f.lower().endswith('.csv')]
            if files:
                # Preferir german_credit_data.csv si existe
                preferred = 'german_credit_data.csv'
                if preferred in files:
                    csv_path = os.path.join(data_dir, preferred)
                else:
                    csv_path = os.path.join(data_dir, files[0])

        if csv_path:
            print(f'Usando dataset encontrado: {csv_path}')
            df = cargar_csv(csv_path)
            df = mapear_columnas_comunes(df)
            if 'target' not in df.columns:
                print('Aviso: no se detectó columna `target`. Se generará `target` sintético más abajo.'
        else:
            # Ningún CSV disponible: generar datos sintéticos como último recurso
            print('No se encontró CSV en `data/`. Generando datos sintéticos de respaldo.')
            df = generar_datos_sinteticos(2000)

    # No se requiere que el dataset tenga income/debt_ratio/credit_score.
    # La pipeline usará columnas disponibles (por ejemplo `loan_amount`, `duration`, `Age`) y
    # generará `target` sintético si no existe.

    # Calcular característica difusa: si el dataset contiene las columnas tradicionales
    if all(c in df.columns for c in ['income', 'debt_ratio', 'credit_score']):
        df['fuzzy_risk'] = lote_puntuaciones_difusas(df)
    else:
        # Intentar calcular fuzzy_risk a partir de monto/duración/edad si están disponibles
        from src.fuzzy_module import lote_riesgo_desde_prestamo
        if any(c in df.columns for c in ['loan_amount', 'Credit amount', 'Duration', 'duration', 'Age',
            df['fuzzy_risk'] = lote_riesgo_desde_prestamo(df)
        else:
            raise ValueError('No hay columnas adecuadas para calcular la característica difusa')

    # Si no existe columna `target`, crear una sintética con una heurística simple
    if 'target' not in df.columns:
        import numpy as np
```

```
        def _sigmoid(x):
            return 1 / (1 + np.exp(-x))

        # Usar Age, loan_amount (Credit amount) y Duration si están disponibles
        age_col = 'Age' if 'Age' in df.columns else ('age' if 'age' in df.columns else None)
        loan_col = 'loan_amount' if 'loan_amount' in df.columns else ('Credit amount' if 'Credit amount'
        dur_col = 'Duration' if 'Duration' in df.columns else ('duration' if 'duration' in df.columns el

        if loan_col is None and dur_col is None and age_col is None:
            # fallback: usar fuzzy_risk
            df['target_prob'] = df['fuzzy_risk']
        else:
            # combinar en una heurística
            coeff_loan = 0.0005 if loan_col else 0.0
            coeff_dur = 0.05 if dur_col else 0.0
            coeff_age = -0.02 if age_col else 0.0

            loan_vals = df[loan_col].astype(float) if loan_col else 0
            dur_vals = df[dur_col].astype(float) if dur_col else 0
            age_vals = df[age_col].astype(float) if age_col else 0

            logit = coeff_loan * loan_vals + coeff_dur * dur_vals + coeff_age * age_vals + 0.5 * df['fuz
            df['target_prob'] = _sigmoid(logit)

        df['target'] = (df['target_prob'] > 0.5).astype(int)

    features_base = ['income', 'debt_ratio', 'credit_score', 'loan_amount', 'employment_length']
    # Filter features that exist
    features_base = [f for f in features_base if f in df.columns]

    results = []

    # Models without fuzzy feature
    X, y = preparar_xy(df, features_base)
    for name in ['logistic', 'random_forest']:
        mname = 'logistic' if name == 'logistic' else 'random_forest'
        metrics, model = entrenar_y_evaluar(X, y, model_name=('logistic' if name == 'logistic' else 'ran
        metrics['with_fuzzy'] = False
        results.append(metrics)

    # Models with fuzzy feature
    Xf = X.copy()
    Xf['fuzzy_risk'] = df['fuzzy_risk']
    for name in ['logistic', 'random_forest']:
        metrics, model = entrenar_y_evaluar(Xf, y, model_name=('logistic' if name == 'logistic' else 'ra
        metrics['with_fuzzy'] = True
        results.append(metrics)

    results_df = pd.DataFrame(results)
    results_df.to_csv('output/results.csv', index=False)

    # Simple plot: ROC placeholder via roc_auc metric bar
    plt.figure(figsize=(8, 5))
    sns.barplot(data=results_df, x='model', y='roc_auc', hue='with_fuzzy')
    plt.title('ROC AUC por modelo (con / sin fuzzy)')
    plt.ylim(0, 1)
    plt.savefig('output/figures/roc_auc_comparison.png')
    print('Resultados guardados en output/results.csv y figuras en output/figures/')


def _map_credit_history_to_score(hist: str) -> float:
    if not hist:
        return 55.0
    hist_low = hist.strip().lower()
    if 'malo' in hist_low or 'poor' in hist_low:
        return 30.0
    if 'regular' in hist_low or 'fair' in hist_low:
        return 55.0
    if 'bueno' in hist_low or 'good' in hist_low:
        return 75.0
    return 55.0


def _risk_label_from_score(score: float) -> str:
    if score < 0.4:
        return 'RIESGO BAJO'
    if score < 0.7:
        return 'RIESGO MEDIO'
    return 'RIESGO ALTO'
def mostrar_evaluacion_solicitante(applicant: dict, df):
    """Muestra por consola una evaluación explicable del solicitante.
```

```
    applicant puede contener: age (años), income_monthly (moneda local), debt_ratio (porcentaje 0-1 o 0-
    credit_history (texto)
    """
    # Normalizar campos
    age = applicant.get('age', None)
    income_monthly = applicant.get('income_monthly', None)
    debt_ratio = applicant.get('debt_ratio', None)
    credit_history = applicant.get('credit_history', None)

    # Convertir income a miles porque las funciones fuzzy esperan "income" en miles
    income_thousands = None
    if income_monthly is not None:
        # si parece estar en unidades (ej S/ 2,800), convertir a miles
        try:
            income_val = float(str(income_monthly).replace(',', '').replace('S/', '').strip())
            income_thousands = income_val / 1000.0
        except Exception:
            income_thousands = None

    # debt_ratio en 0-1
    dr = None
    if debt_ratio is not None:
        try:
            drv = float(str(debt_ratio).replace('%', '').strip())
            if drv > 1:
                dr = drv / 100.0
            else:
                dr = drv
        except Exception:
            dr = None

    # map historial a puntaje
    credit_score = _map_credit_history_to_score(credit_history)

    # Valores por defecto si faltan
    if income_thousands is None:
        income_thousands = 2.8
    if dr is None:
        dr = 0.45
    if age is None:
        age = 30

    # Calcular riesgo difuso (usar función explicativa)
    expl = explicar_riesgo(income_thousands, dr, credit_score)
    fuzzy_score = expl['fuzzy_score']
```

fuzzy_module.py

```
"""
Módulo simple de lógica difusa sin dependencias externas.
Define funciones de membresía y un conjunto de reglas
para calcular un `riesgo_difuso` en [0,1].
"""
from typing import Dict

def memb_triangular(x: float, a: float, b: float, c: float) -> float:
    if x <= a or x >= c:
        return 0.0
    if x == b:
        return 1.0
    if x < b:
        return (x - a) / (b - a)
    return (c - x) / (c - b)

def fuzzificar_ingresos(income: float) -> Dict[str, float]:
    # income in thousands
    return {
        'low': memb_triangular(income, 0, 0, 30),
        'medium': memb_triangular(income, 20, 50, 80),
        'high': memb_triangular(income, 60, 120, 200),
    }

def fuzzificar_ratio_deuda(dr: float) -> Dict[str, float]:
    # dr is ratio 0-1
    return {
        'low': memb_triangular(dr, 0.0, 0.0, 0.2),
        'medium': memb_triangular(dr, 0.1, 0.3, 0.6),
        'high': memb_triangular(dr, 0.4, 0.7, 1.0),
    }
def fuzzificar_puntaje_credito(score: float) -> Dict[str, float]:
```

```
    # score 0-100
    return {
        'poor': memb_triangular(score, 0, 0, 50),
        'fair': memb_triangular(score, 30, 55, 75),
        'good': memb_triangular(score, 60, 80, 100),
    }

def puntuacion_riesgo_difuso(income: float, debt_ratio: float, credit_score: float) -> float:
    inc = fuzzificar_ingresos(income)
    dr = fuzzificar_ratio_deuda(debt_ratio)
    cs = fuzzificar_puntaje_credito(credit_score)

    # Valores lingüísticos mapeados a riesgo numérico: low=0, medium=0.5, high=1
    reglas = []

    reglas.append((min(inc['low'], dr['high']), 1.0))
    reglas.append((cs['poor'], 1.0))
    reglas.append((min(inc['medium'], dr['medium']), 0.5))
    reglas.append((min(inc['high'], cs['good']), 0.0))
    reglas.append((dr['low'], 0.0))
    reglas.append((cs['fair'], 0.5))

    num = 0.0
    den = 0.0
    for fuerza, valor_salida in reglas:
        num += fuerza * valor_salida
        den += fuerza

    if den == 0:
        return 0.5
    return float(num / den)

def lote_puntuaciones_difusas(df):
    # espera columnas: 'income', 'debt_ratio', 'credit_score'
    return df.apply(lambda r: puntuacion_riesgo_difuso(r['income'], r['debt_ratio'], r['credit_score']),


def fuzzificar_monto_prestamo(amount: float) -> Dict[str, float]:
    return {
        'small': memb_triangular(amount, 0, 0, 2000),
        'medium': memb_triangular(amount, 1500, 4000, 8000),
        'large': memb_triangular(amount, 6000, 15000, 30000),
    }


def fuzzificar_duracion(d: float) -> Dict[str, float]:
    return {
        'short': memb_triangular(d, 0, 0, 12),
        'medium': memb_triangular(d, 6, 24, 48),
        'long': memb_triangular(d, 36, 60, 120),
    }


def fuzzificar_edad(age: float) -> Dict[str, float]:
    return {
        'young': memb_triangular(age, 18, 20, 30),
        'adult': memb_triangular(age, 25, 40, 60),
        'senior': memb_triangular(age, 55, 70, 100),
    }


def riesgo_difuso_desde_prestamo(loan_amount: float, duration: float, age: float) -> float:
    la = fuzzificar_monto_prestamo(loan_amount)
    du = fuzzificar_duracion(duration)
    ag = fuzzificar_edad(age)

    reglas = []
    reglas.append((min(la['large'], du['long']), 1.0))
    reglas.append((min(la['small'], du['short']), 0.0))
    reglas.append((min(ag['young'], la['large']), 1.0))
    reglas.append((min(ag['senior'], la['small']), 0.0))
    reglas.append((min(la['medium'], du['medium']), 0.5))

    num = 0.0
    den = 0.0
    for fuerza, valor_salida in reglas:
        num += fuerza * valor_salida
        den += fuerza
    if den == 0:
        return 0.5
    return float(num / den)
```

```
def lote_riesgo_desde_prestamo(df):
    # espera columnas: 'loan_amount', 'Duration' o 'duration' y 'Age' o 'age'
    def _fila(r):
        loan = r.get('loan_amount') if 'loan_amount' in r.index else r.get('Credit amount', 0)
        dur = r.get('duration') if 'duration' in r.index else r.get('Duration', 0)
        age = r.get('age') if 'age' in r.index else r.get('Age', 0)
        return riesgo_difuso_desde_prestamo(float(loan), float(dur), float(age))

    return df.apply(_fila, axis=1)


def explicar_riesgo(income: float, debt_ratio: float, credit_score: float) -> Dict:
    """Devuelve información explicativa: membresías de ingreso y deuda, y reglas activadas.

    income: en miles (como usa el módulo)
    debt_ratio: en 0-1
    credit_score: 0-100
    """
    inc = fuzzificar_ingresos(income)
    dr = fuzzificar_ratio_deuda(debt_ratio)
    cs = fuzzificar_puntaje_credito(credit_score)

    # Reglas (misma lógica que en `puntuacion_riesgo_difuso`)
    reglas = [
        ((min(inc['low'], dr['high'])), 'Si ingreso es BAJO y endeudamiento es ALTO -> riesgo ALTO', 1.0
        ((cs['poor']), 'Si puntaje es POOR -> riesgo ALTO', 1.0),
        ((min(inc['medium'], dr['medium'])), 'Si ingreso es MEDIO y endeudamiento es MEDIO -> riesgo MED
        ((min(inc['high'], cs['good'])), 'Si ingreso es ALTO y puntaje es GOOD -> riesgo BAJO', 0.0),
        ((dr['low']), 'Si endeudamiento es BAJO -> riesgo BAJO', 0.0),
        ((cs['fair']), 'Si puntaje es FAIR -> riesgo MEDIO', 0.5),
    ]

    # Calcular fuerza total y salida ponderada
    num = 0.0
    den = 0.0
    for fuerza, texto, valor_salida in reglas:
        num += fuerza * valor_salida
        den += fuerza

    score = float(num / den) if den != 0 else 0.5

    # Seleccionar reglas activadas (fuerza > 0)
    activadas = []
    for fuerza, texto, valor_salida in reglas:
        if fuerza > 0:
            activadas.append({'regla': texto, 'fuerza': float(fuerza), 'valor_salida': float(valor_salid

    return {
        'income_membership': inc,
        'debt_ratio_membership': dr,
        'credit_score_membership': cs,
        'activated_rules': activadas,
        'fuzzy_score': score,
    }
```

ml_module.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, precision_score, recall_score
from typing import Tuple, Dict

def preparar_xy(df: pd.DataFrame, features: list, target_col: str = 'target'):
    X = df[features].copy()
    y = df[target_col].values
    return X, y


def entrenar_y_evaluar(X, y, model_name: str = 'logistic') -> Dict:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
    if model_name == 'logistic':
        model = LogisticRegression(max_iter=1000)
    else:
        model = RandomForestClassifier(n_estimators=100, random_state=42)

    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    probs = model.predict_proba(X_test)[:, 1] if hasattr(model, 'predict_proba') else preds
```

```
    # Cálculo seguro de métricas cuando y_test contiene una sola clase
    from numpy import unique
    if len(unique(y_test)) > 1:
        roc = float(roc_auc_score(y_test, probs))
        precision = float(precision_score(y_test, preds, zero_division=0))
        recall = float(recall_score(y_test, preds, zero_division=0))
    else:
        roc = float('nan')
        precision = float(precision_score(y_test, preds, zero_division=0))
        recall = float(recall_score(y_test, preds, zero_division=0))

    metrics = {
        'model': model_name,
        'accuracy': float(accuracy_score(y_test, preds)),
        'roc_auc': roc,
        'precision': precision,
        'recall': recall,
    }
    return metrics, model


def prob_a_tres_clases(prob: float) -> Dict[str, float]:
    """Convierte una probabilidad binaria (riesgo) en una distribución suave de tres clases.

    Usamos funciones triangulares centradas en 0 (BAJO), 0.5 (MEDIO), 1 (ALTO).
    """
    def tri(x, a, b, c):
        if x <= a or x >= c:
            return 0.0
        if x == b:
            return 1.0
        if x < b:
            return (x - a) / (b - a)
        return (c - x) / (c - b)

    p_low = tri(prob, 0.0, 0.0, 0.5)
    p_med = tri(prob, 0.0, 0.5, 1.0)
    p_high = tri(prob, 0.5, 1.0, 1.0)

    total = p_low + p_med + p_high
    if total == 0:
        # fallback: asignar según cercanía
        return {'low': max(0.0, 1 - prob), 'medium': 0.0 + (1 - abs(prob - 0.5)), 'high': max(0.0, prob)
    return {'low': float(p_low / total), 'medium': float(p_med / total), 'high': float(p_high / total)}
```

reporting.py

```
import datetime
import os
from typing import Dict
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Image, PageBreak, Table, TableStyle
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.enums import TA_JUSTIFY, TA_CENTER
from reportlab.lib import colors
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm

def generar_informe_html(output_path: str, applicant: Dict, fuzzy: Dict, ml_probs: Dict = None, final_la
    """Genera un informe HTML simple con la evaluación y métricas.

    output_path: ruta del archivo HTML a escribir
    applicant: dict con datos del solicitante
    fuzzy: dict con keys 'fuzzy_score', 'income_membership', 'debt_ratio_membership', 'activated_rules'
    ml_probs: dict con keys 'low','medium','high'
    metrics: dict con métricas delmodelo
    global_dist: dict con distribución global de etiquetas
    figure_path: ruta relativa a una figura para incluir
    """
    titulo = "Informe de Evaluación de Riesgo Crediticio"
    now = datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
    lines = []
    lines.append(f"<html><head><meta charset=\"utf-8\"><title>{titulo}</title></head><body>")
    lines.append(f"<h1>{titulo}</h1>")
    lines.append(f"<p><small>Generado: {now}</small></p>")

    lines.append('<h2>Datos del solicitante</h2>')
    lines.append('<ul>')
    for k, v in applicant.items():
        lines.append(f"<li><strong>{k}:</strong> {v}</li>")
    lines.append('</ul>')
```

```
    lines.append('<h2>Módulo Lógica Difusa</h2>')
    lines.append(f"<p>Score difuso: {fuzzy.get('fuzzy_score', 0):.2f}</p>")
    lines.append('<h3>Membresías</h3>')
    lines.append('<ul>')
    for name, val in fuzzy.get('income_membership', {}).items():
        lines.append(f"<li>Ingreso {name}: {val:.2f}</li>")
    for name, val in fuzzy.get('debt_ratio_membership', {}).items():
        lines.append(f"<li>Endeudamiento {name}: {val:.2f}</li>")
    lines.append('</ul>')
    lines.append('<h3>Reglas activadas</h3>')
    lines.append('<ul>')
    for r in fuzzy.get('activated_rules', []):
        lines.append(f"<li>{r.get('regla')} (fuerza={r.get('fuerza'):.2f})</li>")
    lines.append('</ul>')

    if ml_probs:
        lines.append('<h2>Módulo Machine Learning</h2>')
        lines.append('<ul>')
        lines.append(f"<li>Prob Riesgo Bajo: {ml_probs.get('low',0):.2f}</li>")
        lines.append(f"<li>Prob Riesgo Medio: {ml_probs.get('medium',0):.2f}</li>")
        lines.append(f"<li>Prob Riesgo Alto: {ml_probs.get('high',0):.2f}</li>")
        lines.append('</ul>')

    if final_label:
        lines.append(f"<h2>Decisión Final: {final_label}</h2>")

    if metrics:
        lines.append('<h2>Métricas del Modelo</h2>')
        lines.append('<ul>')
        for k, v in metrics.items():
            lines.append(f"<li>{k}: {v}</li>")
        lines.append('</ul>')

    if global_dist:
        lines.append('<h2>Distribución Global (muestra)</h2>')
        lines.append('<ul>')
        for k, v in global_dist.items():
            lines.append(f"<li>{k}: {v:.2%}</li>")
        lines.append('</ul>')

    if figure_path and os.path.exists(figure_path):
        lines.append('<h2>Figura</h2>')
        lines.append(f'<img src="{figure_path}" alt="Figura" style="max-width:600px;">')

    lines.append('<hr><p>Este informe fue generado automáticamente por el sistema híbrido.</p>')
    lines.append('</body></html>')

    with open(output_path, 'w', encoding='utf-8') as f:
        f.write('\n'.join(lines))

    return output_path


def generar_pdf_desde_html(html_path: str, pdf_path: str) -> bool:
    """Intenta convertir un HTML a PDF. Primero intenta WeasyPrint, luego pdfkit.

    Devuelve True si la conversión fue exitosa.
    """
    # Intentar WeasyPrint
    try:
        from weasyprint import HTML
        HTML(html_path).write_pdf(pdf_path)
        return True
    except Exception:
        pass

    # Intentar pdfkit (requiere wkhtmltopdf instalado en el sistema)
    try:
        import pdfkit
        pdfkit.from_file(html_path, pdf_path)
        return True
    except Exception:
        pass

    return False


def generar_informe_pdf(pdf_path: str, applicant: Dict, fuzzy: Dict, ml_probs: Dict = None, final_label:
    """Genera un PDF directamente (intenta WeasyPrint, luego pdfkit).

    Crea un HTML en memoria con la figura embebida (si existe) y lo convierte.
```

```
Devuelve True si el PDF fue creado.
"""
# Construir HTML en memoria
titulo = "Informe de Evaluación de Riesgo Crediticio"
now = datetime.datetime.now().strftime('%Y-%m-%d %H:%M')
parts = [f"<html><head><meta charset='utf-8'><title>{titulo}</title></head><body>"]
parts.append(f"<h1>{titulo}</h1>")
parts.append(f"<p><small>Generado: {now}</small></p>")
parts.append('<h2>Datos del solicitante</h2><ul>')
for k, v in applicant.items():
    parts.append(f"<li><strong>{k}:</strong> {v}</li>")
parts.append('</ul>')
parts.append('<h2>Módulo Lógica Difusa</h2>')
parts.append(f"<p>Score difuso: {fuzzy.get('fuzzy_score', 0):.2f}</p>")
parts.append('<h3>Membresías</h3><ul>')
for name, val in fuzzy.get('income_membership', {}).items():
    parts.append(f"<li>Ingreso {name}: {val:.2f}</li>")
for name, val in fuzzy.get('debt_ratio_membership', {}).items():
    parts.append(f"<li>Endeudamiento {name}: {val:.2f}</li>")
parts.append('</ul>')
parts.append('<h3>Reglas activadas</h3><ul>')
for r in fuzzy.get('activated_rules', []):
    parts.append(f"<li>{r.get('regla')} (fuerza={r.get('fuerza'):.2f})</li>")
parts.append('</ul>')

if ml_probs:
    parts.append('<h2>Módulo Machine Learning</h2><ul>')
    parts.append(f"<li>Prob Riesgo Bajo: {ml_probs.get('low',0):.2f}</li>")
    parts.append(f"<li>Prob Riesgo Medio: {ml_probs.get('medium',0):.2f}</li>")
    parts.append(f"<li>Prob Riesgo Alto: {ml_probs.get('high',0):.2f}</li>")
    parts.append('</ul>')

if final_label:
    parts.append(f"<h2>Decisión Final: {final_label}</h2>")

if metrics:
    parts.append('<h2>Métricas del Modelo</h2><ul>')
    for k, v in metrics.items():
        parts.append(f"<li>{k}: {v}</li>")
    parts.append('</ul>')

if global_dist:
    parts.append('<h2>Distribución Global (muestra)</h2><ul>')
    for k, v in global_dist.items():
        parts.append(f"<li>{k}: {v:.2%}</li>")
    parts.append('</ul>')

# Incluir figura con ruta absoluta si existe
if figure_path and os.path.exists(figure_path):
    abs_path = os.path.abspath(figure_path)
    parts.append('<h2>Figura</h2>')
    parts.append(f"<img src='file:///{abs_path}' alt='Figura' style='max-width:600px;'>")

parts.append('<hr><p>Informe generado automáticamente por el sistema híbrido.</p></body></html>')
html_str = '\n'.join(parts)

# Intentar WeasyPrint
# Intentar ReportLab (pure Python, no binarios externos)
try:
    from reportlab.lib.pagesizes import A4
    from reportlab.pdfgen import canvas
    from reportlab.lib.utils import ImageReader

    c = canvas.Canvas(pdf_path, pagesize=A4)
    width, height = A4
    text = c.beginText(40, height - 40)
    text.setLeading(14)
    text.setFont('Helvetica-Bold', 14)
    text.textLine(titulo)
    text.setFont('Helvetica', 10)
    text.textLine(f"Generado: {now}")
    text.textLine(' ')
    text.setFont('Helvetica-Bold', 12)
    text.textLine('Datos del solicitante:')
    text.setFont('Helvetica', 10)
    for k, v in applicant.items():
        text.textLine(f"- {k}: {v}")
    text.textLine(' ')
    text.setFont('Helvetica-Bold', 12)
    text.textLine('Módulo Lógica Difusa:')
    text.setFont('Helvetica', 10)
```

```
        text.textLine(f"Score difuso: {fuzzy.get('fuzzy_score',0):.2f}")
        for name, val in fuzzy.get('income_membership', {}).items():
            text.textLine(f"Ingreso {name}: {val:.2f}")
        for name, val in fuzzy.get('debt_ratio_membership', {}).items():
```

web_app.py

```
import os
import io
import joblib
import streamlit as st
import pandas as pd

from src.utils import generar_datos_sinteticos, cargar_csv, mapear_columnas_comunes
from src.fuzzy_module import explicar_riesgo, puntuacion_riesgo_difuso
from src.ml_module import preparar_xy, entrenar_y_evaluar, prob_a_tres_clases
from src.reporting import generar_informe_html


MODEL_PATH = os.path.join('output', 'models', 'model_logistic.pkl')


def cargar_o_entrenar_modelo(df=None):
    os.makedirs('output/models', exist_ok=True)
    if os.path.exists(MODEL_PATH):
        data = joblib.load(MODEL_PATH)
        return data['model'], data['features'], data.get('metrics', {})

    # si no existe, entrenar con df o datos sintéticos
    if df is None:
        df = generar_datos_sinteticos(1000)
    df = mapear_columnas_comunes(df)
    features = [f for f in ['income', 'debt_ratio', 'credit_score'] if f in df.columns]
    if not features:
        return None, None, None
    X, y = preparar_xy(df, features)
    metrics, model = entrenar_y_evaluar(X, y, model_name='logistic')
    joblib.dump({'model': model, 'features': features, 'metrics': metrics}, MODEL_PATH)
    return model, features, metrics


def main():
    st.set_page_config(page_title='Evaluación Riesgo Crediticio', layout='wide')
    st.title('Evaluación de Riesgo Crediticio — Sistema Híbrido (Difuso + ML)')

    with st.sidebar:
        st.header('Entrada del solicitante')
        age = st.number_input('Edad', min_value=18, max_value=100, value=30)
        income = st.number_input('Ingreso mensual (S/)', min_value=0.0, value=2800.0, step=100.0)
        debt_pct = st.slider('Nivel de endeudamiento (%)', 0, 100, 45)
        credit_history = st.selectbox('Historial crediticio', ['Regular', 'Bueno', 'Malo', 'Desconocido'
        uploaded = st.file_uploader('Dataset CSV (opcional) para entrenar / estimar distribución', type=
        if uploaded is not None:
            try:
                df_uploaded = pd.read_csv(uploaded)
            except Exception:
                df_uploaded = None
        else:
            df_uploaded = None

        st.markdown('---')
        if st.button('Reentrenar modelo con dataset cargado / sintético'):
            try:
                df_train = df_uploaded if df_uploaded is not None else generar_datos_sinteticos(2000)
                df_train = mapear_columnas_comunes(df_train)
                features = [f for f in ['income', 'debt_ratio', 'credit_score'] if f in df_train.columns
                if not features:
                    st.warning('No hay características válidas en el dataset para entrenar.')
                    st.markdown('**Columnas detectadas en el dataset:**')
                    st.write(list(df_train.columns))
                    st.markdown('Si tu dataset tiene columnas equivalentes con otros nombres, selecciona
                    income_choice = st.selectbox('Columna para `income` (ingresos mensuales o anuales)',
                    debt_choice = st.selectbox('Columna para `debt_ratio` ó `debt` (deuda total)', ['(ni
                    credit_choice = st.selectbox('Columna para `credit_score` (puntaje)', ['(ninguna)']
                    if st.button('Aplicar mapeo y reentrenar', key='apply_map'):
                        rename_map = {}
                        if income_choice != '(ninguna)':
                            rename_map[income_choice] = 'income'
                        if credit_choice != '(ninguna)':
                            rename_map[credit_choice] = 'credit_score'
```

```
                    # Si el usuario indicó una columna que representa deuda y existe columna de ingr
                    if debt_choice != '(ninguna)':
                        if debt_choice in df_train.columns and income_choice != '(ninguna)' and inco
                            try:
                                df_train['debt_ratio'] = df_train[debt_choice].astype(float) / df_tr
                            except Exception:
                                # fallback: renombrar si tiene formato ratio
                                rename_map[debt_choice] = 'debt_ratio'
                        else:
                            rename_map[debt_choice] = 'debt_ratio'

                    if rename_map:
                        df_train = df_train.rename(columns=rename_map)

                    features = [f for f in ['income', 'debt_ratio', 'credit_score'] if f in df_train
                    if not features:
                        st.error('No se pudieron mapear columnas a las características requeridas. F
                    else:
                        X, y = preparar_xy(df_train, features)
                        metrics, model = entrenar_y_evaluar(X, y, model_name='logistic')
                        joblib.dump({'model': model, 'features': features, 'metrics': metrics}, MODE
                        st.success(f'Modelo entrenado y guardado. Métricas: {metrics}')
                else:
                    X, y = preparar_xy(df_train, features)
                    metrics, model = entrenar_y_evaluar(X, y, model_name='logistic')
                    joblib.dump({'model': model, 'features': features, 'metrics': metrics}, MODEL_PATH)
                    st.success(f'Modelo entrenado y guardado. Métricas: {metrics}')
            except Exception as e:
                st.error(f'Error al reentrenar: {e}')

    # Cargar modelo (si existe) o entrenar uno rápido
    model, features, metrics = cargar_o_entrenar_modelo(df_uploaded)

    st.header('Resultados')
    col1, col2 = st.columns(2)

    # Preparar applicant
    income_thousands = income / 1000.0
    dr = debt_pct / 100.0

    # map historial a puntaje
    credit_score = 55.0
    if credit_history:
        lowhist = credit_history.strip().lower()
        if 'malo' in lowhist or 'poor' in lowhist:
            credit_score = 30.0
        elif 'bueno' in lowhist or 'good' in lowhist:
            credit_score = 75.0
        else:
            credit_score = 55.0

    expl = explicar_riesgo(income_thousands, dr, credit_score)

    with col1:
        st.subheader('Lógica Difusa')
        st.write('Score difuso:', f"{expl['fuzzy_score']:.2f}")
        st.write('Membresías (ingreso):', expl['income_membership'])
        st.write('Membresías (endeudamiento):', expl['debt_ratio_membership'])
        if expl['activated_rules']:
            st.markdown('**Reglas activadas (top 3)**')
            for r in sorted(expl['activated_rules'], key=lambda x: x['fuerza'], reverse=True)[:3]:
                st.write(f"{r['regla']} (fuerza={r['fuerza']:.2f})")

    with col2:
        st.subheader('Machine Learning')
        if model is None:
            st.warning('No hay modelo ML disponible.')
        else:
            # predecir probabilidad
            row = pd.DataFrame([{ 'income': income_thousands if 'income' in features else 0, 'debt_ratio
            try:
                prob = float(model.predict_proba(row[features])[:,1][0]) if hasattr(model, 'predict_prob
                probs3 = prob_a_tres_clases(prob)
                st.write('Probabilidad (riesgo binario):', f"{prob:.2f}")
                st.write('Distribución en 3 clases:')
                st.write(probs3)
                # mostrar barras
                st.bar_chart(pd.DataFrame({'prob':[probs3['low'], probs3['medium'], probs3['high']]}, in
            except Exception as e:
                st.error(f'Error al predecir: {e}')
    # Decisión híbrida
```

```
    st.header('Decisión Híbrida')
    if model is None:
        final_label = 'RIESGO BAJO' if expl['fuzzy_score'] < 0.4 else ('RIESGO MEDIO' if expl['fuzzy_sco
        st.write('Resultado final (solo difuso):', final_label)
    else:
        ml_class = None
        try:
            ml_class = max(probs3.items(), key=lambda x: x[1])[0]
            ml_label = 'RIESGO BAJO' if ml_class == 'low' else ('RIESGO MEDIO' if ml_class == 'medium' e
        except Exception:
            ml_label = 'N/A'

        fuzzy_label = 'RIESGO BAJO' if expl['fuzzy_score'] < 0.4 else ('RIESGO MEDIO' if expl['fuzzy_sco
        if ml_label == fuzzy_label:
            final_label = ml_label
            justification = 'Ambos módulos coinciden.'
        else:
            perf = metrics.get('roc_auc') if metrics and not pd.isna(metrics.get('roc_auc')) else metric
            prefer_ml = (perf is not None) and (perf >= 0.65)
            if prefer_ml:
                final_label = ml_label
                justification = f'Se prioriza ML (performance={perf:.2f}).'
            else:
                final_label = fuzzy_label
                justification = f'Se prioriza lógica difusa (performance ML={perf}).'

        st.write('Lógica Difusa:', fuzzy_label, f"(score={expl['fuzzy_score']:.2f})")
        st.write('Machine Learning:', ml_label)
        st.write('Resultado Final (Híbrido):', final_label)
        st.write('Justificación:', justification)

    # Descargar informe (PDF) en un solo paso
    try:
        os.makedirs('output/reports', exist_ok=True)
        figpath = 'output/figures/distribucion_riesgo_fuzzy.png'
        # intentar crear figura si no existe
        try:
            import matplotlib.pyplot as _plt
            if not os.path.exists(figpath):
                sample = (df_uploaded if df_uploaded is not None else generar_datos_sinteticos(500)).cop
                sample = mapear_columnas_comunes(sample)
                sample['fuzzy_score'] = sample.apply(lambda r: puntuacion_riesgo_difuso(r.get('income',0
                sample['fuzzy_label'] = sample['fuzzy_score'].apply(lambda s: 'RIESGO BAJO' if s<0.4 els
                _plt.figure(figsize=(5,5))
                sample['fuzzy_label'].value_counts().plot.pie(autopct='%1.1f%%', ylabel='')
```

utils.py

```
import numpy as np
import pandas as pd
from typing import Tuple

def generar_datos_sinteticos(n: int = 2000, random_state: int = 42) -> pd.DataFrame:
    rng = np.random.RandomState(random_state)
    # ingresos en miles
    ingreso = rng.normal(50, 20, size=n).clip(5, 200)
    # ratio de deuda 0-1
    ratio_deuda = rng.beta(2, 5, size=n)
    # puntaje crediticio 0-100
    puntaje_credito = (rng.normal(60, 15, size=n)).clip(0, 100)
    # edad y antigüedad laboral
    edad = rng.normal(40, 12, size=n).clip(18, 90)
    antiguedad = rng.poisson(5, size=n)
    monto_prestamo = rng.normal(20, 10, size=n).clip(1, 150)
    # probabilidad base de riesgo (verdad sintética)
    risk_logit = (
        -0.03 * ingreso
        + 3.5 * ratio_deuda
        -0.02 * puntaje_credito
        + 0.01 * monto_prestamo
        -0.01 * antiguedad
        + rng.normal(0, 1, size=n)
    )
    prob = 1 / (1 + np.exp(-risk_logit))
    objetivo = (prob > 0.5).astype(int)

    df = pd.DataFrame({
        'income': ingreso,
        'debt_ratio': ratio_deuda,
        'credit_score': puntaje_credito,
```

```
        'age': edad,
        'employment_length': antiguedad,
        'loan_amount': monto_prestamo,
        'target': objetivo,
    })
    return df


def cargar_csv(path: str):
    df = pd.read_csv(path)
    return df


def mapear_columnas_comunes(df: pd.DataFrame) -> pd.DataFrame:
    """Mapea nombres de columnas comunes (inglés/español) a los nombres usados por la pipeline.

    Objetivo: normalizar columnas a `income`, `debt_ratio`, `credit_score`, `loan_amount`,
    `employment_length`, `target` cuando sea posible.
    """
    cols = {c.lower(): c for c in df.columns}

    # Sinónimos para cada campo
    synonyms = {
        'income': ['income', 'ingreso', 'monthly_income', 'salary', 'ingresos'],
        'debt_ratio': ['debt_ratio', 'dti', 'debttoincome', 'debt_to_income', 'ratio_deuda'],
        'credit_score': ['credit_score', 'score', 'fico', 'puntaje', 'credit_score_value'],
        'loan_amount': ['loan_amount', 'loan', 'monto', 'amount', 'credit amount', 'credit_amount', 'cre
        'duration': ['duration', 'loan_duration', 'term', 'duracion'],
        'employment_length': ['employment_length', 'emp_len', 'antiguedad', 'tiempo_empleo'],
        'target': ['target', 'default', 'loan_status', 'label', 'y']
    }

    rename_map = {}
    for target_col, possibles in synonyms.items():
        for p in possibles:
            if p in cols:
                rename_map[cols[p]] = target_col
                break

    # Si existe columna 'debt' y 'income' podemos crear debt_ratio
    if 'debt' in cols and 'income' in cols and 'debt_ratio' not in rename_map:
        df['debt_ratio'] = df[cols['debt']] / df[cols['income']].replace({0: 1})
        rename_map['debt_ratio'] = 'debt_ratio'

    if rename_map:
        df = df.rename(columns=rename_map)
    return df
```

14.2 Dataset (muestra de columnas)

Columnas del dataset:
- Id
- Age
- Sex
- Job
- Housing
- Saving accounts
- Checking account
- Credit amount
- Duration
- Purpose

Tamaño: 51 registros