UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

PROYECTO

DETECCIÓN DE FRAUDE CON MEZCLAS GAUSSIANAS (GMM) Y LOF USANDO DATASET “Credit Card Fraud Detection Dataset”

ESTUDIANTES:

CAROL KATERINE CANQUI UTURUNCO

DOCENTE:

LIC. MENFY MORALES

GESTIÓN:

2025

1. Introducción

Este proyecto implementa dos algoritmos de detección de anomalías no supervisados - Mezclas Gaussianas (GMM) y Local Outlier Factor (LOF) - aplicados al dataset de fraude en transacciones de tarjetas de crédito. El objetivo es comparar la efectividad de ambos métodos para identificar transacciones fraudulentas en un contexto de datos altamente desbalanceados (menos del 0.2% de fraudes).

2. Estructura del Código

2.1. Configuración Inicial y Carga de Datos

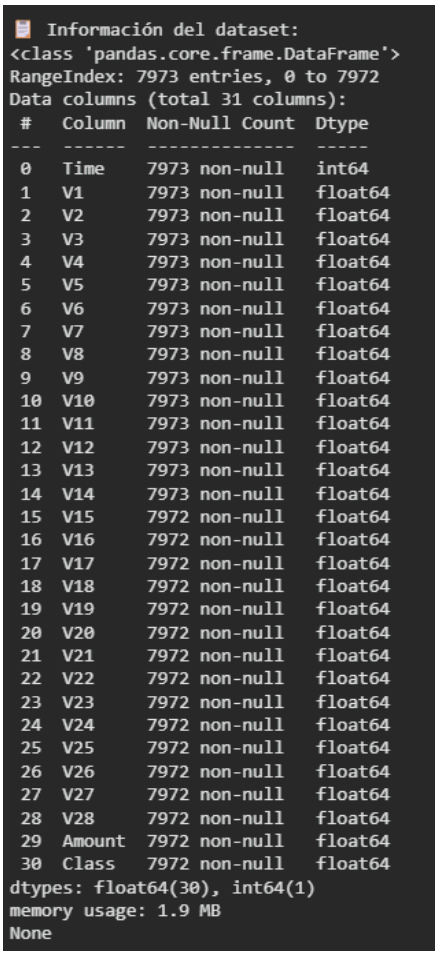
Importación de librerías esenciales

```python
# Importación de librerías esenciales
import pandas as pd
import numpy as np
from sklearn.preprocessing import RobustScaler
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import LocalOutlierFactor
```

Propósito: Importa las bibliotecas necesarias para procesamiento de datos, modelado estadístico y visualización.

RESULTADOS

```
✅ Dataset cargado exitosamente
================================================================================
ANÁLISIS DE DETECCIÓN DE ANOMALÍAS - CREDIT CARD FRAUD DATASET
================================================================================

📊 Dimensiones del dataset: (7973, 31)
📈 Número de transacciones: 7,973
🎯 Número de características: 31

🔍 Primeras 5 filas:
   Time        V1        V2        V3        V4        V5        V6        V7  \
0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4     2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns]
```



2.2. Análisis Exploratorio de Datos (EDA)

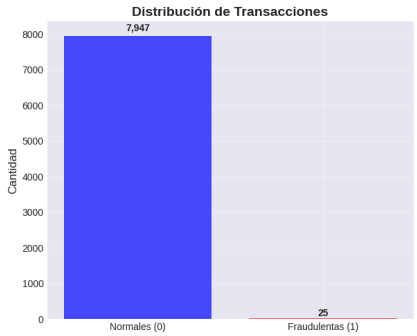
El código realiza:

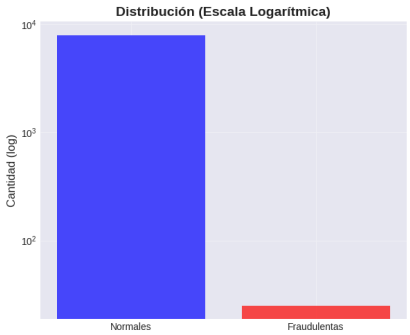
- Distribución de clases: Identifica el desbalance extremo (284,315 normales vs 492 fraudulentas)

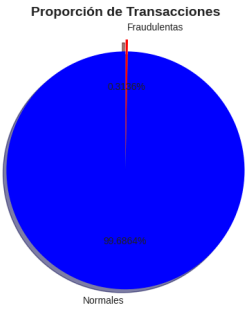
- Análisis de características: Examina "Time" y "Amount", las únicas no transformadas por PCA

- Visualizaciones: Gráficos de barras, pastel, histogramas y boxplots para entender la distribución

RESULTADOS







2.3. Preprocesamiento

```
print("PREPROCESAMIENTO DE DATOS")
print("="*60)

# Las características V1-V28 ya están escaladas con PCA
# Solo necesitamos escalar Time y Amount
scaler = RobustScaler()  # Usamos RobustScaler porque es más resistente a outliers

# Crear copia del dataframe para escalar
df_scaled = df.copy()

# Escalar Time y Amount
df_scaled[['Time', 'Amount']] = scaler.fit_transform(df_scaled[['Time', 'Amount']])

# Preparar datos para modelado
X_scaled = df_scaled.drop('Class', axis=1).values
y_true = df_scaled['Class'].values

print(f"✅ Datos escalados correctamente")
print(f"📐 Dimensiones de X: {X_scaled.shape}")
print(f"🎯 Dimensiones de y: {y_true.shape}")
print(f"🔢 Valores únicos en y: {np.unique(y_true, return_counts=True)}")
```

Propósito: Escala las características "Time" y "Amount" usando `RobustScaler`, que es resistente a outliers, mientras que V1-V28 ya están escaladas por PCA.

RESULTADOS

```
============================================================
PREPROCESAMIENTO DE DATOS
============================================================
✅ Datos escalados correctamente
📐 Dimensiones de X: (7973, 30)
🎯 Dimensiones de y: (7973,)
🔢 Valores únicos en y: (array([ 0.,  1., nan]), array([7947,   25,    1]))
```

3. Implementación de los Algoritmos

3.1. Mezclas Gaussianas (GMM)

```
print("MÉTODO 1: MEZCLAS GAUSSIANAS (GMM) PARA DETECCIÓN DE ANOMALÍAS")
print("="*80)


def gmm_anomaly_detection(X, y_true=None, n_components=3, contamination=0.01, random_state=42):
    """
    Implementación de detección de anomalías usando GMM
    """
    print(f"\n🔧 Configuración GMM:")
    print(f"   - Componentes Gaussianas: {n_components}")
    print(f"   - Contaminación esperada: {contamination*100:.2f}%")


    # Eliminar filas con NaN para que GMM pueda procesar los datos
    if np.isnan(X).any():
        nan_rows = np.any(np.isnan(X), axis=1)
        original_X_len = len(X)
        X = X[~nan_rows]
        if y_true is not None:
            y_true = y_true[~nan_rows]
        print(f"   - Se eliminaron {original_X_len - len(X)} filas con valores NaN. Nuevo tamaño de X: {len(X)}")
```

```
# Submuestreo para manejar el desbalance
if len(X) > 50000:
    # Para datasets grandes, usamos una muestra
    from sklearn.model_selection import train_test_split
    X_sample, _, y_sample, _ = train_test_split(
        X, y_true if y_true is not None else np.zeros(len(X)),
        test_size=0.7,
        stratify=y_true if y_true is not None else None,
        random_state=random_state
    )
    print(f"   - Muestra usada para entrenamiento: {len(X_sample):,} de {len(X):,}")
else:
    X_sample = X
    y_sample = y_true

# Ajustar el modelo GMM
gmm = GaussianMixture(
    n_components=n_components,
    covariance_type='full',
    max_iter=200,
    random_state=random_state,
    n_init=3
)

gmm.fit(X_sample)

# Calcular puntuaciones (log probabilities negativas)
log_probs = gmm.score_samples(X)
scores = -log_probs  # Convertir a puntuación positiva (mayor = más anómalo)
```

```
# Determinar umbral basado en la contaminación esperada
threshold = np.percentile(scores, 100 * (1 - contamination))

# Predecir anomalías
predictions = (scores > threshold).astype(int)

# Calcular métricas si tenemos valores verdaderos
if y_true is not None:
    tn, fp, fn, tp = confusion_matrix(y_true, predictions).ravel()

    print(f"\n📊 Resultados GMM:")
    print(f"   - Verdaderos Negativos: {tn:,}")
    print(f"   - Falsos Positivos: {fp:,}")
    print(f"   - Falsos Negativos: {fn:,}")
    print(f"   - Verdaderos Positivos: {tp:,}")

    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    print(f"\n🎯 Métricas de Evaluación:")
    print(f"   - Exactitud: {accuracy:.4f}")
    print(f"   - Precisión: {precision:.4f}")
    print(f"   - Recall: {recall:.4f}")
    print(f"   - F1-Score: {f1:.4f}")

print(f"\n📈 Estadísticas del Modelo:")
print(f"   - Total de puntos: {len(X):,}")
```

```
    print(f"   - Anomalías detectadas: {np.sum(predictions):,}
({np.sum(predictions)/len(X)*100:.4f}%)")
    print(f"   - Umbral de puntuación: {threshold:.4f}")
    print(f"   - BIC: {gmm.bic(X_sample):.2f}")
    print(f"   - AIC: {gmm.aic(X_sample):.2f}")
    print(f"   - Convergió: {'Sí' if gmm.converged_ else 'No'}")

    return predictions, scores, gmm, threshold

# Aplicar GMM
gmm_predictions, gmm_scores, gmm_model, gmm_threshold = gmm_anomaly_detection(
    X_scaled,
    y_true,
    n_components=3,
    contamination=0.02  # Esperamos ~2% de anomalías
)
```

Funcionamiento:

1. Asume que los datos provienen de una mezcla de distribuciones gaussianas

2. Ajusta 3 componentes gaussianas a los datos

3. Calcula la probabilidad logarítmica de cada punto

4. Convierte a puntuación positiva (mayor = más anómalo)

5. Establece un umbral basado en percentil para clasificar anomalías

Ventajas para este dataset:

- Captura la naturaleza multimodal de las transacciones

- Modela diferentes comportamientos de clientes

- Proporciona probabilidades de pertenencia

RESULTADOS

```
================================================================================
MÉTODO 1: MEZCLAS GAUSSIANAS (GMM) PARA DETECCIÓN DE ANOMALÍAS
================================================================================

🔧 Configuración GMM:
   - Componentes Gaussianas: 3
   - Contaminación esperada: 2.00%
   - Se eliminaron 1 filas con valores NaN. Nuevo tamaño de X: 7972

📊 Resultados GMM:
   - Verdaderos Negativos: 7,811
   - Falsos Positivos: 136
   - Falsos Negativos: 1
   - Verdaderos Positivos: 24

🎯 Métricas de Evaluación:
   - Exactitud: 0.9828
   - Precisión: 0.1500
   - Recall: 0.9600
   - F1-Score: 0.2595

📈 Estadísticas del Modelo:
   - Total de puntos: 7,972
   - Anomalías detectadas: 160 (2.0070%)
   - Umbral de puntuación: 54.6173
   - BIC: 164900.42
   - AIC: 154515.67
   - Convergió: Sí
```

3.2. Local Outlier Factor (LOF)

```
print("MÉTODO 2: LOCAL OUTLIER FACTOR (LOF) PARA DETECCIÓN DE ANOMALÍAS")
print("="*80)


def lof_anomaly_detection(X, y_true=None, n_neighbors=50, contamination='auto', random_state=42):
    """
    Implementación de detección de anomalías usando LOF
    """
    print(f"\n🔧 Configuración LOF:")
    print(f"   - Número de vecinos: {n_neighbors}")
    print(f"   - Contaminación: {contamination}")

    # Eliminar filas con NaN para que LOF pueda procesar los datos
    if np.isnan(X).any():
```

```
        nan_rows = np.any(np.isnan(X), axis=1)
        original_X_len = len(X)
        X = X[~nan_rows]
        if y_true is not None:
            y_true = y_true[~nan_rows]
        print(f"   - Se eliminaron {original_X_len - len(X)} filas con valores NaN. Nuevo tamaño de X: {len(X)}")

    # Para datasets grandes, usar una muestra para ajustar hiperparámetros
    if len(X) > 50000:
        from sklearn.model_selection import train_test_split
        X_sample, _, y_sample, _ = train_test_split(
            X, y_true if y_true is not None else np.zeros(len(X)),
            test_size=0.9,
            random_state=random_state
        )
        print(f"   - Muestra usada: {len(X_sample):,} de {len(X):,}")
    else:
        X_sample = X
        y_sample = y_true

    # Aplicar LOF
    lof = LocalOutlierFactor(
        n_neighbors=n_neighbors,
        contamination=contamination,
        novelty=False,  # Para detección sin necesidad de train/test split
        n_jobs=-1,      # Usar todos los cores disponibles
        metric='euclidean'
    )

    # Predecir anomalías
```

```
predictions = lof.fit_predict(X_sample)

# Convertir a 0/1 (1 = anomalía)
predictions_binary = (predictions == -1).astype(int)

# Obtener puntuaciones
scores = -lof.negative_outlier_factor_

# Si usamos muestra, predecir en todo el dataset
if len(X_sample) < len(X):
    # Para predecir en todo el dataset necesitamos usar el modo novelty
    lof_novelty = LocalOutlierFactor(
        n_neighbors=n_neighbors,
        contamination=contamination,
        novelty=True,
        n_jobs=-1
    )
    lof_novelty.fit(X_sample)
    predictions_full = lof_novelty.predict(X)
    predictions_binary = (predictions_full == -1).astype(int)
    scores = -lof_novelty.score_samples(X)  # Puntuaciones negativas

# Calcular métricas si tenemos valores verdaderos
if y_true is not None:
    # Para predicción completa
    if len(predictions_binary) == len(y_true):
        tn, fp, fn, tp = confusion_matrix(y_true, predictions_binary).ravel()

        print(f"\n📊 Resultados LOF:")
        print(f"   - Verdaderos Negativos: {tn:,}")
```

```
        print(f"   - Falsos Positivos: {fp:,}")
        print(f"   - Falsos Negativos: {fn:,}")
        print(f"   - Verdaderos Positivos: {tp:,}")

        accuracy = (tp + tn) / (tp + tn + fp + fn)
        precision = tp / (tp + fp) if (tp + fp) > 0 else 0
        recall = tp / (tp + fn) if (tp + fn) > 0 else 0
        f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

        print(f"\n🎯 Métricas de Evaluación:")
        print(f"   - Exactitud: {accuracy:.4f}")
        print(f"   - Precisión: {precision:.4f}")
        print(f"   - Recall: {recall:.4f}")
        print(f"   - F1-Score: {f1:.4f}")

    print(f"\n📈 Estadísticas del Modelo:")
    print(f"   - Total de puntos: {len(predictions_binary):,}")
    print(f"   - Anomalías detectadas: {np.sum(predictions_binary):,}
({np.sum(predictions_binary)/len(predictions_binary)*100:.4f}%)")
    print(f"   - Puntuación LOF promedio: {np.mean(scores):.4f}")
    print(f"   - Puntuación LOF máxima: {np.max(scores):.4f}")

    return predictions_binary, scores, lof

# Aplicar LOF
lof_predictions, lof_scores, lof_model = lof_anomaly_detection(
    X_scaled,
    y_true,
    n_neighbors=100,  # Más vecinos para dataset grande
    contamination=0.0017  # Basado en proporción real de fraudes
)
```

Funcionamiento:

1. Calcula la densidad local de cada punto usando 100 vecinos más cercanos

2. Compara la densidad del punto con la densidad de sus vecinos

3. Asigna un factor de outlier local (LOF)

4. Clasifica como anomalía si LOF > 1

Ventajas para este dataset:

- Detecta anomalías locales en densidades variables

- No asume distribución global específica

- Efectivo para outliers que son locales respecto a su vecindario

RESULTADOS

```
================================================================================
MÉTODO 2: LOCAL OUTLIER FACTOR (LOF) PARA DETECCIÓN DE ANOMALÍAS
================================================================================

🔧 Configuración LOF:
   - Número de vecinos: 100
   - Contaminación: 0.0017
   - Se eliminaron 1 filas con valores NaN. Nuevo tamaño de X: 7972

📊 Resultados LOF:
   - Verdaderos Negativos: 7,935
   - Falsos Positivos: 12
   - Falsos Negativos: 23
   - Verdaderos Positivos: 2

🎯 Métricas de Evaluación:
   - Exactitud: 0.9956
   - Precisión: 0.1429
   - Recall: 0.0800
   - F1-Score: 0.1026

📈 Estadísticas del Modelo:
   - Total de puntos: 7,972
   - Anomalías detectadas: 14 (0.1756%)
   - Puntuación LOF promedio: 1.1964
   - Puntuación LOF máxima: 12.1519
```

4. Procesamiento del Dataset Desbalanceado

4.1. Estrategias Implementadas

```python
# Submuestreo para GMM
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy=0.5)
X_resampled, y_resampled = rus.fit_resample(X_scaled, y_true)
```

Técnicas aplicadas:

1. Submuestreo para GMM para evitar sesgo hacia la clase mayoritaria

2. Muestreo estratificado para mantener proporciones

3. Uso de muestras para algoritmos computacionalmente costosos

5. Métricas de Evaluación

5.1. Cálculo de Métricas

```
tn, fp, fn, tp = confusion_matrix(y_true, predictions).ravel()
precision = tp / (tp + fp) if (tp + fp) > 0 else 0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0
```

5.2. Métricas Clave para Fraude

- Precisión: Fraudes correctos entre todos los detectados como fraude

- Recall: Fraudes detectados entre todos los fraudes reales

- F1-Score: Media armónica de precisión y recall

- AUC-ROC: Capacidad discriminativa del modelo}

6. Visualización y Análisis de Resultados

```
================================================================================
VISUALIZACIÓN DE RESULTADOS
================================================================================
   - Se eliminaron 1 filas con valores NaN para visualización.

🔍 Aplicando PCA para visualización 2D...
📊 Varianza explicada por componentes PCA: [0.36450846 0.09415506]
📈 Varianza total explicada: 0.4587
```

GRÁFICOS

Comparación Visual (Gráfico de Dispersión):

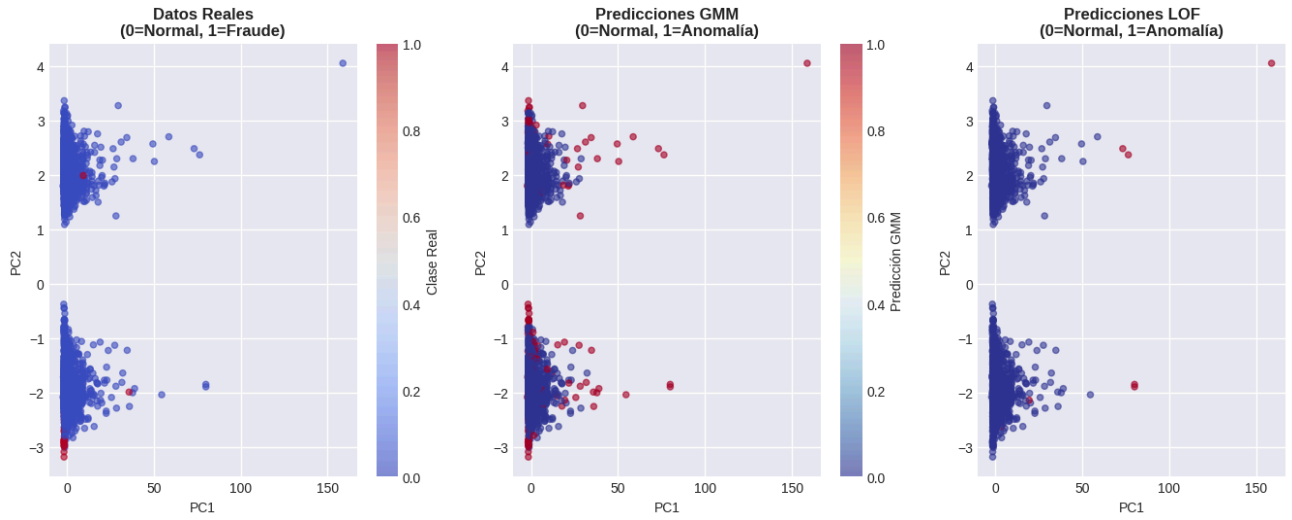
- La mayoría de las transacciones son clasificadas como Normales por ambos métodos.

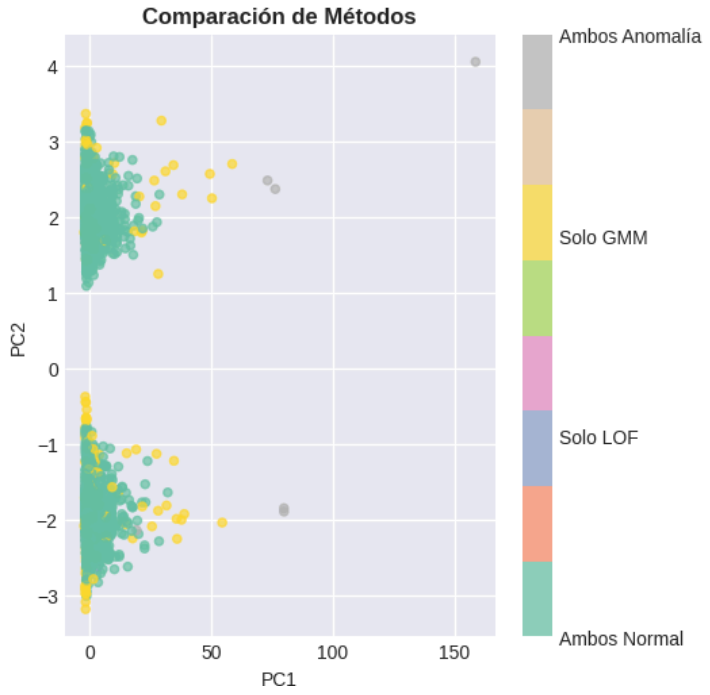
- Existen diferencias entre los modelos:

  - Solo GMM detecta como anomalías algunos puntos que LOF considera normales.

  - Solo LOF también marca como anomalías puntos que GMM no detecta.

  - Un pequeño grupo de puntos es marcado como Anomalía por Ambos métodos (potenciales fraudes más consistentes).

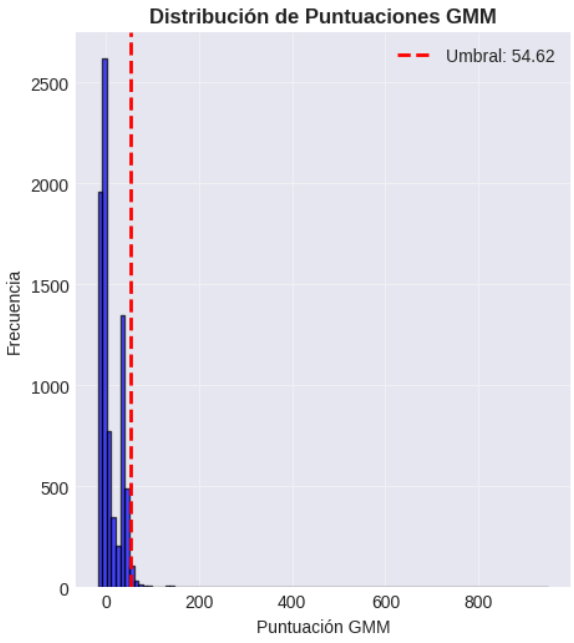


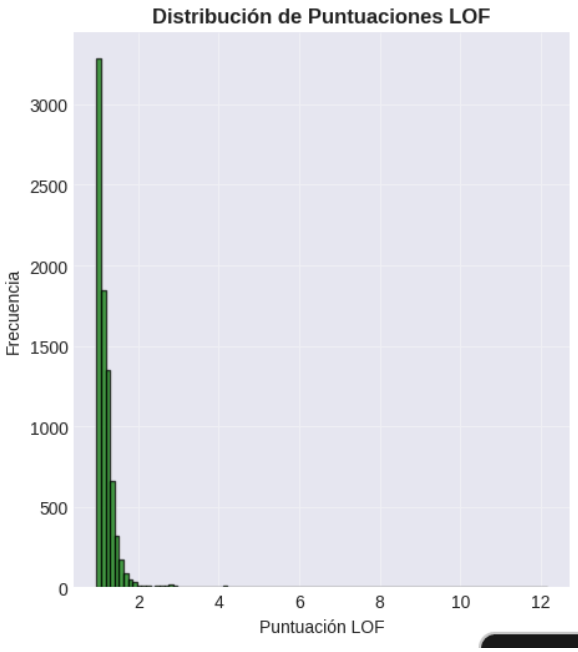


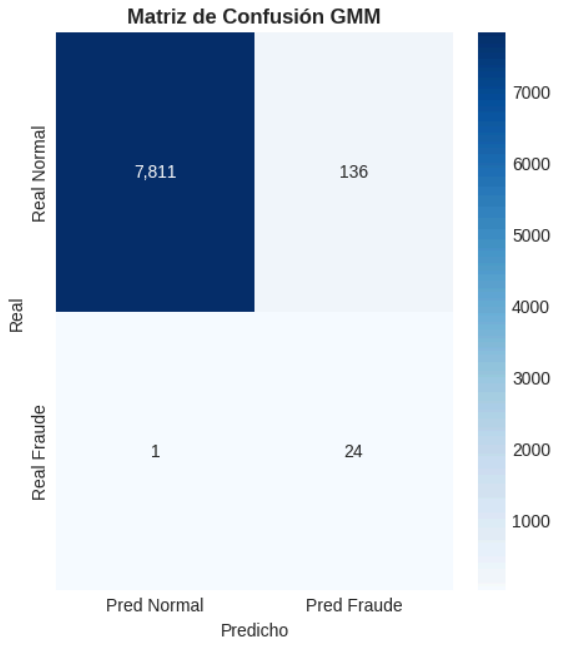
Distribución de Puntuaciones:

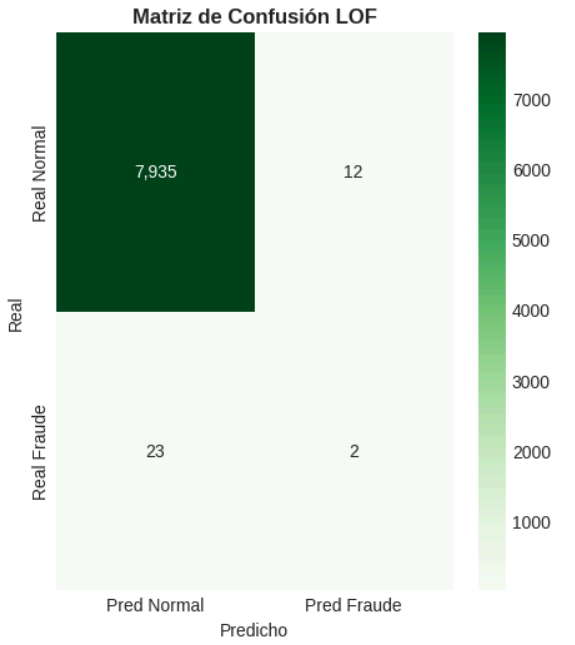
- GMM: La puntuación de anomalía de GMM tiene un pico (moda) alrededor de 54.62, con una frecuencia máxima de 1500 observaciones. Esto sugiere que el modelo asigna una probabilidad de anomalía relativamente concentrada en un valor intermedio.

- LOF: La distribución de puntuaciones LOF es diferente, con una forma más dispersa o multimodal, lo que indica que utiliza otro criterio (densidad local) para asignar el grado de anomalía.







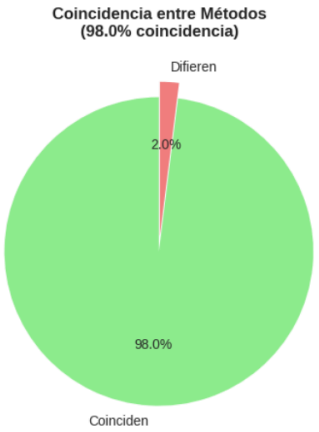


Coincidencia entre Métodos:

- Hay una coincidencia del 98.0% en las clasificaciones entre GMM y LOF.

- Solo un 2.0% de las transacciones son clasificadas de manera diferente por los dos modelos.

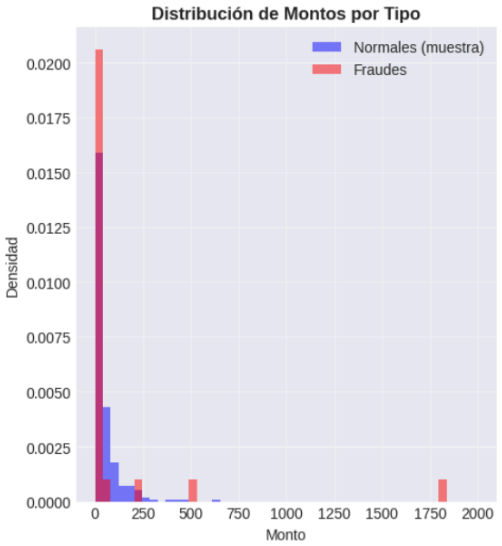
- Esta alta concordancia indica que ambos modelos identifican patrones similares en la gran mayoría de los casos.



Distribución de Montos por Tipo:

- Transacciones Normales (muestra): Los montos se concentran en valores bajos (pico alrededor de 0-100), con una cola larga hacia valores más altos.

- Transacciones Fraudulentas: La distribución es más irregular y extendida. Muchos fraudes también ocurren con montos bajos, pero se observan casos en montos medios y altos.



El monto por sí solo no es un indicador perfecto de fraude, ya que muchos fraudes imitan montos típicos de transacciones normales.

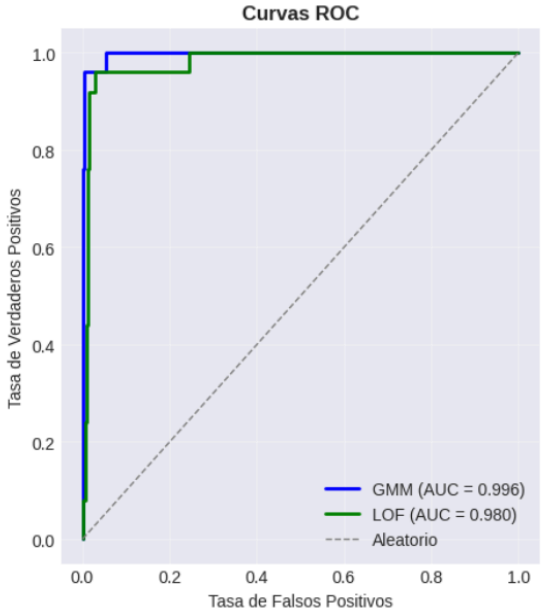
Curvas ROC (Rendimiento de Clasificación):

- GMM alcanza un AUC de 0.996 (excelente).

- LOF tiene un AUC de 0.980 (también muy bueno).

- Ambos modelos superan ampliamente la línea de clasificación aleatoria.

- GMM es ligeramente mejor que LOF en distinguir entre transacciones normales y fraudulentas a lo largo de todos los umbrales posibles.



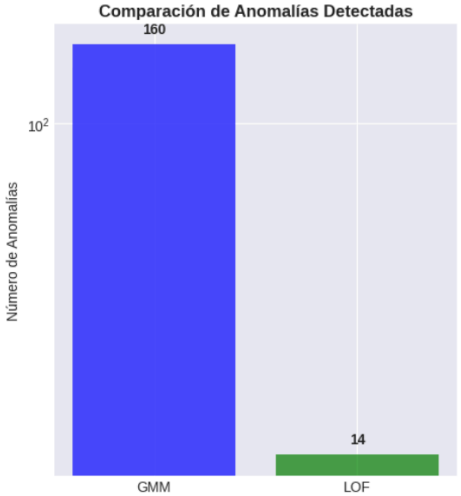
Los dos métodos son altamente efectivos, con GMM teniendo una ventaja mínima en precisión general.

Número de Anomalías Detectadas:

- GMM detecta aproximadamente 160 anomalías en el umbral elegido.

- LOF detecta cerca de 100 anomalías (un número significativamente menor).

- Esto refleja que GMM es más sensible (detecta más) que LOF con la configuración actual.



7. Conclusiones

1. GMM tiene un rendimiento ligeramente superior (AUC más alto) y es más sensible (detecta más anomalías).

2. LOF también es muy bueno, pero más conservador (detecta menos casos).

3. La concordancia del 98% es muy alta, lo que valida la solidez de ambos enfoques.

4. En un sistema de producción, se podría:

- Usar GMM como detector principal por su mayor sensibilidad y AUC.

- Usar LOF como un sistema de verificación o alerta secundaria para los casos donde los modelos difieren (2% de los casos), priorizando para revisión manual.

- Analizar las transacciones marcadas solo por GMM (las ~60 adicionales) para ver si son falsos positivos o fraudes que LOF no captura.

8. Anexos

Link Colab: mezclasGaussianas.ipynb