Informe proyecto final programación: Clasificación de Fraude de Tarjetas

Samuel Ceballos Gómez – Ing. Mecatrónica

---

1. Planteamiento del problema

El objetivo de este trabajo es **predecir transacciones fraudulentas** en un histórico de tarjetas de crédito. Para ello, utilizamos un conjunto de datos etiquetado con transacciones legítimas y fraudulentas, y entrenamos dos modelos supervisados distintos para comparar su desempeño:

1. **Regresión Logística**
2. **Random Forest**

Queremos determinar cuál de estos dos algoritmos ofrece mejores métricas de clasificación (AUC, precisión, recall y F1-score) y si sus predicciones coinciden en gran medida (mediante el coeficiente Kappa de Cohen).

---

2. Dataset y preprocesado

2.1. Descripción inicial del dataset

- El dataset original proviene de Kaggle ("Credit Card Fraud Detection") y contiene **284 807 registros** y **31 variables** numéricas (todas anonimizado como "V1"…"V28", más "Time", "Amount" y la etiqueta "Class").
- Al inspeccionar con df.info() y df.isnull().sum(), constatamos que **no hay valores faltantes** en ninguna columna.

2.2. Simulación e imputación del 5 % de valores faltantes

- Para cumplir la consigna, simulamos el 5 % de valores faltantes seleccionando aleatoriamente celdas del DataFrame (con semilla fija random_state=42).
- Una vez asignados los NaN al 5 % de celdas, utilizamos SimpleImputer(strategy='median') para imputar cada columna numérica con su mediana respectiva.
- Después de la imputación, confirmamos con df_imputed.isnull().sum() que **no quedan valores faltantes**.

2.3. Balanceo de clases

- En el dataset original, la clase "no fraude" (Class = 0) era abrumadoramente mayoritaria.
- Extraímos todas las filas donde Class == 1 (fraudes) y las filas donde Class == 0 (no fraudes). El total de fraudes resultó ser 462 registros.

- Para balancear, submuestreamos aleatoriamente 462 registros de la clase “no fraude” (con semilla fija random_state=42) y los concatenamos con los 462 fraudes.
- El **dataset final balanceado** tiene **924 registros** y **32 columnas** (después de renombrar “Amount_scaled” a “Amount”).
- Confirmamos la distribución de clases con:

Class
0 462
1 462
Name: Class, dtype: int64

2.4. Escalado de variable Amount

- Aplicamos StandardScaler() únicamente sobre la columna original “Amount” y la renombramos a “Amount”.
- Las 30 variables “V1”…“V28” y la nueva “Amount” estandarizada conforman las 31 características finales (más la etiqueta “Class”).

2.5. Resumen de shapes

- **Antes de preprocesar:** 284 807 filas × 31 columnas.
- **Después de imputar y balancear:** 924 filas × 32 columnas (incluida “Class”).

---

3. Modelos entrenados

A partir del dataset preprocesado (924 × 32), dividimos en **80 % entrenamiento (739 registros)** y **20 % prueba (185 registros)** de manera estratificada.

3.1. Modelo 1: Regresión Logística

- **GridSearchCV**
  - Rango de hiperparámetros:
    - C: [0.01, 0.1, 1, 10, 100]
    - penalty: [‘l1’, ‘l2’] (con solver='liblinear', max_iter=1000)
  - Validación cruzada 5-fold, optimizando la métrica **ROC-AUC**.
  - **Mejores parámetros encontrados:**

C = 1
penalty = 'l2'

-
  -
  - **Mejor AUC en validación cruzada:** 0.9029
- **Entrenamiento final**
  - Se ajusta el modelo con C=1 y penalty='l2' sobre los 739 registros de entrenamiento.
  - **Predicciones en prueba (185 registros)** con best_log.predict y best_log.predict_proba.

- **Métricas en conjunto de prueba**
  - **AUC**: 0.9501
  - **Precisión (precision_score)**: 0.9515
  - **Recall (recall_score)**: 0.8529
  - **F1-score (f1_score)**: 0.8991
  - **Matriz de confusión**:

[[89 4]
 [13 79]]

-
  -
    - 89 verdaderos negativos (no fraude correctamente identificados)
    - 4 falsos positivos (no fraude clasificado como fraude)
    - 13 falsos negativos (fraude clasificado como no fraude)
    - 79 verdaderos positivos (fraudes correctamente identificados)
- **Curva ROC**
  - La curva ROC muestra el trade-off entre TPR (True Positive Rate) y FPR (False Positive Rate).
  - El área bajo la curva (AUC) en prueba coincide con 0.9501, lo que refleja muy buen desempeño discriminativo.
- **Curva de aprendizaje**
  - Se graficaron las puntuaciones de AUC en entrenamiento vs. validación al variar el tamaño del conjunto de entrenamiento (de 10 % a 100 %).
  - No se observa gap excesivo: ambas curvas convergen cerca de AUC ≈ 0.90 en validación, lo que sugiere que el modelo **no sobreajusta demasiado** y que agregar más datos podría estabilizar aún más la métrica.

3.2. Modelo 2: Random Forest

- **GridSearchCV**
  - Rango de hiperparámetros:
    - n_estimators: [50, 100, 200]
    - max_depth: [None, 10, 20]
    - min_samples_split: [2, 5]
    - min_samples_leaf: [1, 2]
    - random_state fijo en 42
  - Validación cruzada 5-fold, optimizando **ROC-AUC**.
  - **Mejores parámetros encontrados:**

n_estimators = 200
max_depth = 10
min_samples_split = 2
min_samples_leaf = 1

-
  -
  - **Mejor AUC en validación cruzada (RF):** 0.9123

- **Entrenamiento final**
  - Se ajusta RandomForestClassifier(n_estimators=200, max_depth=10, min_samples_split=2, min_samples_leaf=1, random_state=42) sobre los 739 registros de entrenamiento.
  - Se generan predicciones en el conjunto de prueba (185 registros).
- **Métricas en conjunto de prueba**
  - **AUC**: 0.9735
  - **Precisión (precision_score)**: 0.9873
  - **Recall (recall_score)**: 0.8478
  - **F1-score (f1_score)**: 0.9123
  - **Matriz de confusión**:

[[92 1]
 [14 78]]

  -
  -
    - 92 verdaderos negativos
    - 1 falso positivo
    - 14 falsos negativos
    - 78 verdaderos positivos
- **Curva ROC**
  - El AUC en test (0.9735) muestra que Random Forest **mejora la capacidad discriminativa** frente a Regresión Logística.
  - La curva ROC está más cerca de la esquina superior izquierda.
- **Curva de aprendizaje**
  - Al analizar la AUC en entrenamiento vs. validación según el tamaño del conjunto de entrenamiento, se observa que:
    - Con pocos datos, la AUC de entrenamiento es muy alta (~0.99) y la de validación un poco menor (~0.90),
    - Conforme aumenta la cantidad de datos, ambas curvas convergen cerca de 0.97 en validación, lo que sugiere **ligero sobreajuste inicial**, pero que se mitiga con más datos.

---

4. Resultados y comparación

A continuación, se recogen las métricas clave de ambos modelos evaluados sobre el mismo conjunto de prueba (185 registros):

Métrica	Regresión Logística	Random Forest
AUC	0.9501	0.9735
Precisión	0.9515	0.9873
Recall	0.8529	0.8478
F1-score	0.8991	0.9123

- **AUC**: Random Forest obtiene 0.9735, superando el 0.9501 de regresión logística.

- **Precisión**: Random Forest (0.9873) también es superior a la de regresión logística (0.9515).
- **Recall**: La regresión logística alcanza 0.8529, ligeramente por encima del 0.8478 de Random Forest; esto indica que la Regresión Logística detecta un poco más falsos positivos (menor precisión) a cambio de captar más fraudes verdaderos (mayor recall).
- **F1-score**: Random Forest alcanza 0.9123 frente a 0.8991 de logística, reflejando un mejor equilibrio general entre precisión y recall.

Matriz de confusión (resumen)

- **Regresión Logística** (185 registros):

[[89 4]
[13 79]]

-
- **Random Forest** (185 registros):

[[92 1]
[14 78]]

-
- Observamos que, en prueba, Random Forest comete menos falsos positivos (solo 1 vs. 4) pero un ligero aumento de falsos negativos (14 vs. 13) en comparación con la regresión logística.

4.1. Coeficiente Kappa de Cohen

- Calculamos el **coeficiente Kappa** entre las predicciones binarias de ambos modelos (y_pred_log vs. y_pred_rf).
- El resultado fue **0.8902**, lo que indica un **alto grado de acuerdo** entre las dos técnicas (un valor cercano a 1 implica coincidencia casi total).

---

5. Conclusión

1. **Desempeño global**
    - Ambos modelos muestran muy buen desempeño para la detección de fraude en tarjetas, con AUC superiores a 0.95.
    - **Random Forest** destaca por su AUC (0.9735) y precisión (0.9873) más elevados que la Regresión Logística.
    - La **Regresión Logística** ofrece mayor interpretabilidad (coeficientes lineales que se pueden examinar), y su **recall** es marginalmente mejor (0.8529 vs. 0.8478), lo que puede interesar en entornos donde la prioridad sea capturar la mayor cantidad de fraudes posibles, aún a costa de más falsos positivos.
2. **Ventajas y desventajas**
    - **Random Forest**
        - Mejor AUC y precisión.
        - – Modelo menos interpretable (“caja negra”).

        - – Requiere ajustar más hiperparámetros y puede ser más costoso en tiempo de entrenamiento/inferencia.
    - **Regresión Logística**
        -
            - Interpretabilidad directa (coeficientes explican la importancia de cada variable).
            - Más rápido de entrenar y más sencillo de implementar.
        - – Ligera pérdida de AUC y precisión frente a Random Forest.
3. **Posibles mejoras futuras**
    - Evaluar otros algoritmos como **XGBoost** o **LightGBM**, que suelen obtener AUC aún más altos en este tipo de datos según mi investigación previa.
    - Explorar técnicas de **ensembles** (stacking) que integren Regresión Logística, Random Forest y otros clasificadores.
    - Analizar la importancia de variables (feature importance) del Random Forest para identificar patrones específicos de fraude.
    - Validar el modelo en un conjunto de datos más grande y en tiempo real, así como ajustar umbrales de decisión (threshold) según el costo asociado a falsos positivos vs. falsos negativos en el negocio.

1. Dataset de credit card fraud

1. **Kaggle – Credit Card Fraud Detection**
    - URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
    - Descripción: Página oficial del dataset, donde se puede descargar el archivo creditcard.csv. Incluye información básica sobre las variables (p. ej. las transformaciones PCA “V1…V28”), el origen de los datos y la licencia, así como estadísticas generales.

2. **Artículo asociado al dataset**
    - Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, Gianluca Bontempi, “Calibrating Probability with Undersampling for Unbalanced Classification,” 2015.
    - Enlace (PDF): https://arxiv.org/abs/1605.12543
    - Descripción: Paper que describe el uso de este dataset para la detección de fraude en transacciones. Explica la motivación, el preprocesado original (PCA de variables), la distribución altamente desbalanceada y algunas técnicas de evaluación para problemas de desbalanceo.

---

2. Exploración de datos (EDA)

1. **Pandas Documentation: Getting Started**
   - URL: https://pandas.pydata.org/docs/getting_started/index.html
   - Descripción: Guía rápida de Pandas para entender los métodos DataFrame.info(), DataFrame.describe(), isnull(), value_counts(), etc.

2. **Seaborn Documentation: Visualización Estadística en Python**
   - URL: https://seaborn.pydata.org/
   - Secciones consultadas:
     - distributions (para sns.histplot)
     - heatmap (para la función sns.heatmap)
     - boxplot (para sns.boxplot)
     -
3. **Matplotlib Documentation**
   - URL: https://matplotlib.org/stable/contents.html
   - Secciones consultadas:
     - pyplot (para plt.figure(), plt.plot(), plt.xlabel(), plt.ylabel(), etc.)

4. **Ejemplo de EDA en credit card fraud (blog/tutorial)**
   - "Credit Card Fraud Detection: EDA & Preprocessing" (Kaggle Kernel)
   - URL: https://www.kaggle.com/code/arjuns975/credit-card-fraud-detection-eda-preprocessing
   - Descripción: Notebook de ejemplo donde se muestran análisis exploratorios como histograma de Amount, conteo de clases, heatmap de correlaciones y detección de outliers.

---

3. Preprocesado de datos

1. **scikit-learn Documentation: Imputación de valores faltantes**
   - URL: https://scikit-learn.org/stable/modules/impute.html
   - Sección consultada: sklearn.impute.SimpleImputer (estrategia median)
   - Descripción: Explica cómo usar SimpleImputer para reemplazar valores faltantes con la mediana (u otras estrategias).
2. **scikit-learn Documentation: Feature Scaling**
   - URL: https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling
   - Sección consultada: sklearn.preprocessing.StandardScaler
   - Descripción: Muestra cómo estandarizar variables numéricas (media = 0, desviación estándar = 1).
3. **scikit-learn Documentation: train_test_split**
   - URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

- Descripción: Función para dividir un dataset en subconjuntos de entrenamiento y prueba (parámetro stratify para mantener la proporción de clases).

4. **Tutorial sobre balanceo de clases**
   - "Undersampling for Imbalanced Classes" (blog post)
   - URL: https://machinelearningmastery.com/undersampling-for-imbalanced-classification/
   - Descripción: Explica por qué y cómo submuestrear la clase mayoritaria para balancear un dataset altamente desbalanceado.

---

4. Modelo 1: Regresión Logística

1. **scikit-learn Documentation: LogisticRegression**
   - URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
   - Descripción: Detalla los parámetros de LogisticRegression (por ejemplo, solver='liblinear', penalty='l1' o 'l2', C, max_iter, etc.).
2. **scikit-learn Documentation: Búsqueda de Hiperparámetros (GridSearchCV)**
   - URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
   - Descripción: Muestra cómo configurar GridSearchCV con el parámetro param_grid y cómo especificar la métrica a optimizar (scoring='roc_auc', cv=5, etc.).
3. **scikit-learn Documentation: Métricas de clasificación**
   - classification_report
     - URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
   - roc_auc_score
     - URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html
   - confusion_matrix
     - URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
   - Descripción: Cálculo de métricas (precisión, recall, F1-score, AUC) y generación de matriz de confusión.
4. **scikit-learn Tutorial: Curvas ROC**
   - URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

- Descripción: Ejemplo que muestra cómo obtener fpr, tpr y graficar la curva ROC con roc_curve y auc.

5. **scikit-learn Tutorial: Curvas de aprendizaje**
   - URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
   - Sección consultada: Uso de learning_curve() para calcular puntajes de entrenamiento y validación a diferentes tamaños de muestra.
6. **Artículo sobre Regresión Logística en casos desbalanceados**
   - "Logistic Regression for Imbalanced Data" (Towards Data Science)
   - URL: https://towardsdatascience.com/logistic-regression-for-imbalanced-datasets-51bc6838dc8c
   - Descripción: Conceptos sobre cómo ajustar C y el impacto de penalty en datos desbalanceados.

---

5. Modelo 2: Random Forest

1. **scikit-learn Documentation: RandomForestClassifier**
   - URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
   - Descripción: Explica los parámetros n_estimators, max_depth, min_samples_split, min_samples_leaf, random_state, etc.
2. **scikit-learn Documentation: GridSearchCV**
   - (Ver sección en Modelo 1, misma URL).
   - Descripción: Configuración de param_grid específico para Random Forest.
3. **scikit-learn Tutorial: Importancia de variables en Random Forest**
   - URL: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html
   - Descripción: Ejemplo de cómo extraer feature_importances_ (si se desea profundizar en la explicación de variables).
4. **scikit-learn Tutorial: Curvas de aprendizaje**
   - (Ver sección en Modelo 1, misma URL).
   - Descripción: Cálculo de puntuaciones de entrenamiento vs. validación para Random Forest.
5. **Blog / Tutorial: Random Forest en detección de fraude**
   - "Credit Card Fraud Detection using Random Forest"
   - URL: https://medium.com/analytics-vidhya/credit-card-fraud-detection-using-random-forest-b6ea19bde4a2
   - Descripción: Explica la implementación paso a paso de Random Forest sobre el mismo dataset, incluyendo preprocesado y evaluación de métricas.

---

6. Comparación de modelos y métricas adicionales

1. **scikit-learn Documentation: cohen_kappa_score**
    - URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html
    - Descripción: Función para calcular el coeficiente Kappa de Cohen, que evalúa el grado de acuerdo entre dos clasificadores (o entre clasificador y etiqueta "verdadera").
2. **Artículo sobre Kappa de Cohen**
    - "Cohen's Kappa: Measuring Agreement" (Practical Data Science)
    - URL: https://practicaldatascience.co.uk/machine-learning/how-to-calculate-cohens-kappa-measure-agreement-between-two-labelers-using-scikit-learn
    - Descripción: Explica la interpretación de valores de Kappa (cercano a 0 = acuerdo aleatorio, cercano a 1 = acuerdo perfecto) y ejemplos prácticos.
3. **Tutorial / Blog: Comparación de modelos de clasificación**
    - "Comparing Classifiers: Logistic Regression vs Random Forest"
    - URL: https://analyticsindiamag.com/logistic-regression-vs-random-forest/
    - Descripción: Análisis de diferencias teóricas entre ambos modelos y recomendaciones de uso según interpretabilidad o desempeño.

---

7. Herramientas generales y utilidades

1. **Google Colab**
    - Google Colab: https://colab.research.google.com/
    - Documentación: https://research.google.com/colaboratory/faq.html
2. **GitHub**
    - Para alojar el código y los notebooks: https://github.com/
    - Repositorio específico (ProyectoFinal_Progra): https://github.com/OferTrabajo/ProyectoFinal_Progra