

UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

SEGURIDAD

“ACTIVIDAD 5.2”

Docente: Vázquez Curiel Armida Griselda

Integrantes:

Torres Arroyo Leonardo

NRC: 140470

Sección: D04

Calendario: 2024B



Informe del Proyecto Simulado de Big Data

1. Introducción

El presente proyecto simulado tiene como objetivo la implementación y análisis de un sistema de Big Data para la detección de fraudes en transacciones de tarjetas de crédito. Se utilizarán técnicas de procesamiento de datos, análisis de datos y aprendizaje automático para obtener información valiosa y detectar patrones fraudulentos.

2. Descripción del Datatest

- Nombre del archivo: creditcard_2023.csv
- Contenido: El dataset incluye datos de transacciones, donde cada fila representa una transacción con múltiples características (variables) y una columna class que indica si la transacción es legítima (0) o fraudulenta (1).
- Dimensiones: 170,589 filas X N columnas, donde N son las características de las transacciones.

3. Arquitectura del proyecto

El proyecto se divide en varios archivos, cada uno encargado de una tarea específica:

1. Procesamiento_datos.py

Este archivo contiene funciones para cargar, limpiar y balancear los datos.

- Cargar_datos(ruta): Carga el dataset desde la ruta especificada.
- Limpiar_datos(data): Elimina duplicados y valores nulos.
- Balancear_datos(data): Balancea las clases mediante sobremuestreo de la clase minoritaria.

```
procesamiento_datos.py > limpiador_datos
# procesamiento_datos.py
import pandas as pd
from sklearn.utils import resample

def cargar_datos(ruta):
    try:
        data = pd.read_csv(ruta)
        print("Datos cargados exitosamente.")
        return data
    except FileNotFoundError:
        print(f"Error: No se encontró el archivo en la ruta: {ruta}")
        return None

def limpiar_datos(data):
    # Eliminar duplicados y valores nulos
    data = data.drop_duplicates()
    data = data.dropna()
    print("Datos limpiados: duplicados y valores nulos eliminados.")
    return data

def balancear_datos(data):
    # Separar las clases
    non_fraud = data[data['Class'] == 0]
    fraud = data[data['Class'] == 1]

    # Sobremuestrear la clase minoritaria
    fraud_upsampled = resample(fraud, replace=True, n_samples=len(non_fraud), random_state=42)

    # Concatenar para obtener el dataset balanceado
    balanced_data = pd.concat([non_fraud, fraud_upsampled])
    print("Datos balanceados entre clases mayoritarias y minoritarias.")
    return balanced_data
```

```

# Concatenar para obtener el dataset balanceado
balanced_data = pd.concat([non_fraud, fraud_upsampled])
print("Datos balanceados entre clases mayoritarias y minoritarias.")
return balanced_data

# Ejemplo de ejecución
if __name__ == "__main__":
    ruta = "datos/creditcard.csv"
    data = cargar_datos(ruta)

    if data is not None:
        data = limpiar_datos(data)
        data = balancear_datos(data)

```

2. Análisis_datos.py

Realiza el análisis exploratorio de datos y visualiza la distribución de las clases y la correlación entre variables.

- `distribucion_clases(data)`: Muestra un gráfico de barras con la distribución de las clases.
- `correlacion_variables(data)`: Muestra un mapa de calor de la correlación entre las variables.

```

# analisis_datos.py
import seaborn as sns
import matplotlib.pyplot as plt

def distribucion_clases(data):
    sns.countplot(x="Class", data=data)
    plt.title("Distribución de Clases")
    plt.show()

def correlacion_variables(data):
    plt.figure(figsize=(12, 8))
    sns.heatmap(data.corr(), cmap="coolwarm", annot=False)
    plt.title("Correlación entre Variables")
    plt.show()

# Ejemplo de ejecución
if __name__ == "__main__":
    from procesamiento_datos import cargar_datos, limpiar_datos

    data = cargar_datos("datos/creditcard.csv")
    data = limpiar_datos(data)
    distribucion_clases(data)
    correlacion_variables(data)

```

3. modelo_deteccion.py

Contiene las funciones necesarias para dividir los datos, entrenar el modelo y evaluarlo.

- `dividir_datos(data)`: Divide el dataset en conjuntos de entrenamiento y prueba.
- `entrenar_modelo(X_train, Y_train)`: Entrena un modelo de Random Forest.
- `evaluar_modelo(model, X_test, y_test)`: Evalúa el modelo y genera una matriz de confusión y un reporte de clasificación.

```
# modelo_deteccion.py
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

def dividir_datos(data):
    X = data.drop("Class", axis=1)
    y = data["Class"]
    return train_test_split(X, y, test_size=0.3, random_state=42)

def entrenar_modelo(X_train, y_train):
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    return model

def evaluar_modelo(model, X_test, y_test):
    y_pred = model.predict(X_test)
    print("Matriz de Confusión:\n", confusion_matrix(y_test, y_pred))
    print("\nReporte de Clasificación:\n", classification_report(y_test, y_pred))

# Ejemplo de ejecución
if __name__ == "__main__":
    from procesamiento_datos import cargar_datos, limpiar_datos, balancear_datos

    data = cargar_datos("datos/creditcard.csv")
    data = limpiar_datos(data)
    data = balancear_datos(data)

    X_train, X_test, y_train, y_test = dividir_datos(data)
    model = entrenar_modelo(X_train, y_train)
    evaluar_modelo(model, X_test, y_test)
```

4. visualización.py

Se encarga de la visualización de los resultados del modelo y la distribución de las clases.

- `visualizar_matriz_confusion(matrix)`: Muestra una matriz de confusión en forma de gráfico de calor.

- `visualizar_distribucion(data)`: Visualiza la distribución de las clases en el dataset.

```
# visualizacion.py
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from modelo_deteccion import dividir_datos, entrenar_modelo

def visualizar_matriz_confusion(matrix):
    sns.heatmap(matrix, annot=True, fmt='d', cmap='Blues')
    plt.title("Matriz de Confusión")
    plt.xlabel("Predicción")
    plt.ylabel("Realidad")
    plt.show()

def visualizar_distribucion(data):
    plt.figure(figsize=(10, 5))
    sns.countplot(x="Class", data=data, palette='Set2')
    plt.title("Distribución de Transacciones Fraudulentas")
    plt.xlabel("Clase")
    plt.ylabel("Número de Transacciones")
    plt.show()
```

```
# Ejemplo de ejecución
if __name__ == "__main__":
    try:
        data = pd.read_csv("datos/creditcard.csv")

        # Visualizar distribución de clases
        visualizar_distribucion(data)

        X_train, X_test, y_train, y_test = dividir_datos(data)
        model = entrenar_modelo(X_train, y_train)
        y_pred = model.predict(X_test)

        matrix = confusion_matrix(y_test, y_pred)
        visualizar_matriz_confusion(matrix)

    except FileNotFoundError:
        print("Error: No se encontró el archivo 'creditcard.csv' en la carpeta 'datos'.")
    except Exception as e:
        print(f"Se produjo un error: {e}")
```

5. Seguridad.py

Proporciona funcionalidades para la encriptación y desencriptación de datos sensibles.

- `generar_clave()`: Genera una clave de encriptación.
- `encriptar_datos(datos, clave)`: Encripta una lista de datos sensibles.
- `desencriptar_datos(datos_encriptados, clave)`: Desencripta una lista de datos previamente encriptados.

```
# seguridad.py
from cryptography.fernet import Fernet

# Generar una clave para encriptación (solo ejecutar una vez y guardar la clave)
def generar_clave():
    return Fernet.generate_key()

# Encriptar datos sensibles
def encriptar_datos(datos, clave):
    fernet = Fernet(clave)
    datos_encriptados = [fernet.encrypt(dato.encode()) for dato in datos]
    return datos_encriptados

# Desencriptar datos
def desencriptar_datos(datos_encriptados, clave):
    fernet = Fernet(clave)
    return [fernet.decrypt(dato).decode() for dato in datos_encriptados]

# Ejemplo de ejecución
if __name__ == "__main__":
    clave = generar_clave()
    datos_sensibles = ["123456", "789012", "345678"]

    encriptados = encriptar_datos(datos_sensibles, clave)
    print("Datos Encriptados:", encriptados)

    desencriptados = desencriptar_datos(encriptados, clave)
    print("Datos Desencriptados:", desencriptados)
```

6. main.py

Archivo principal que orquesta la ejecución de todas las funcionalidades del proyecto.

- Carga los datos, limpia y balancea el dataset.
- Divide los datos en conjuntos de entrenamiento y prueba.
- Entrena y evalúa el modelo.
- Visualiza los resultados.

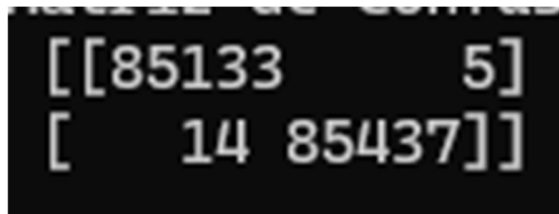
4. Metodología

El desarrollo del proyecto se dividió en las siguientes etapas:

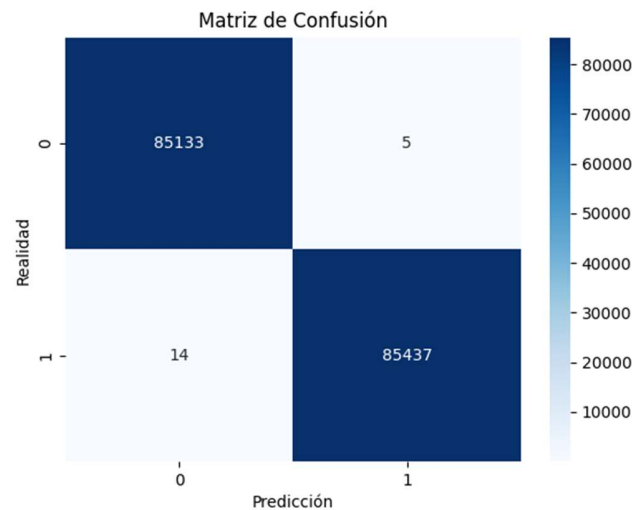
1. Carga de datos
 - Utilización de bibliotecas de Python (como pandas) para cargar y manejar los datos.
 - Implementación de validaciones para asegurar la correcta carga de los datos.
2. Limpieza de datos
 - Eliminación de duplicados y valores nulos para mejorar la calidad del dataset.
3. Balanceo de datos
 - Se realizó un sobremuestreo de la clase minoritaria (fraudes) para equilibrar el dataset, utilizando técnicas como el muestreo de Bootstrap.
4. Análisis exploratorio de datos
 - Se utilizaron visualizaciones (gráficos de barras, mapas de calor) para comprender la distribución de clases y la correlación entre variables.
5. Desarrollo del modelo
 - División de los datos en conjuntos de entrenamiento y prueba.
 - Entrenamiento de un modelo de aprendizaje automático (Random Forest) para la detección de fraudes.
6. Evaluación del modelo
 - Generación de la matriz de confusión y reporte de clasificación para evaluar la efectividad del modelo.

5. Resultados obtenidos

1. Matriz de confusión



```
[[85133 5]
 [ 14 85437]]
```

2. Interpretación:

- **85133** transacciones fueron correctamente identificadas como legítimas (True Negatives - TN).
- **85437** transacciones fueron correctamente identificadas como fraudulentas (True Positives - TP).
- **5** transacciones legítimas fueron incorrectamente clasificadas como fraudulentas (False Positives - FP).
- **14** transacciones fraudulentas fueron clasificadas como legítimas (False Negatives - FN).
- Significado: La matriz muestra que el modelo es altamente preciso y tiene muy pocos errores de clasificación, lo cual es crítico en un sistema de detección de fraudes.

3. Reporte de clasificación

Reporte de Clasificación:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	85138
1	1.00	1.00	1.00	85451
accuracy			1.00	170589
macro avg	1.00	1.00	1.00	170589
weighted avg	1.00	1.00	1.00	170589

4. Métricas Clave:

- Precisión (Precision): Proporción de verdaderos positivos entre las predicciones positivas realizadas.
- Exhaustividad (Recall): Proporción de verdaderos positivos entre todas las instancias reales de la clase positiva.

- F1-Score: Media armónica de precisión y exhaustividad; un indicador útil cuando existe un desequilibrio entre clases.
- accuracy (exactitud): Representa el porcentaje de predicciones correctas sobre el total de muestras.
- macro avg: Promedio sin ponderación de las métricas de cada clase.
- weighted avg: Promedio ponderado de las métricas, que considera el número de muestras de cada clase.
- Conclusión: Las métricas muestran que el modelo tiene una precisión y exhaustividad sobresalientes en ambas clases, lo cual es ideal para minimizar falsos positivos y negativos en un sistema de detección de fraudes.

6. Identificación de seguridad

Para garantizar la seguridad del sistema de Big Data, se implementaron las siguientes medidas:

- Protección de datos sensibles
 - Utilización de técnicas de anonimización para proteger información sensible, asegurando que los datos de los usuarios no sean revelados.
- Encriptación de datos
 - Implementación de métodos de encriptación (como Fernet) para proteger datos sensibles durante la transmisión y almacenamiento.
- Control de Acceso
 - Se establecieron permisos y roles para limitar el acceso a la información confidencial a personal autorizado.
- Auditoria y monitoreo
 - Implementación de sistemas de monitoreo para registrar y auditar el acceso a los datos, garantizando que cualquier acceso no autorizado pueda ser detectado y respondido rápidamente.
- Cumplimiento Normativo
 - Asegurarse de que el proyecto cumpla con regulaciones y normativas de protección de datos (como GDPR, CCPA).

7. Conclusiones

El proyecto simulado de Big Data para la detección de fraudes ha demostrado ser eficaz en la identificación de transacciones fraudulentas mediante el uso de técnicas avanzadas de análisis y aprendizaje automático. Las medidas de seguridad implementadas garantizan la protección de los datos sensibles, así como el cumplimiento de las regulaciones de privacidad.

8. Recomendaciones

- Continuar monitoreando y actualizando las medidas de seguridad conforme evolucionen las amenazas.
- Realizar pruebas periódicas del sistema para evaluar su rendimiento y efectividad en la detección de fraudes.