Actividad 13: Programando Random Forest en Python

Gerardo Enrique Torres Flores 2064063

30 de marzo de 2025

1. Introducción

Random Forest es un tipo de ensamble en Machine Learning en donde se **combinan diversos árboles de decisión**, y la salida de cada uno se cuenta como **"un voto"**. La opción más votada será la respuesta final del Bosque Aleatorio. Al igual que el árbol de decisión, Random Forest es un modelo de aprendizaje supervisado utilizado para tareas de clasificación (y también de regresión).

1. Funciona de la siguiente manera:

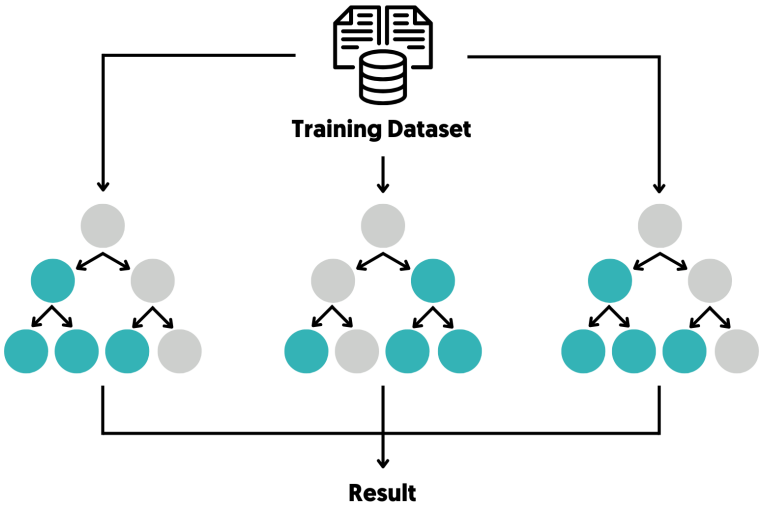
- Se seleccionan k features (columnas) de las m totales (siendo $k < m$) y se crea un árbol de decisión utilizando esas k características.
- Se generan n árboles variando siempre la cantidad de k features y, en algunos casos, variando la cantidad de muestras que se pasan a cada árbol (esto se conoce como "bootstrap sample").
- A cada uno de los n árboles se le pide realizar la misma clasificación y se almacenan sus salidas.
- Se calculan los votos obtenidos para cada clase y se considera como clasificación final la opción más votada.

2. Ventajas

- Funciona bien incluso sin un ajuste exhaustivo de hiperparámetros.
- Es efectivo tanto para problemas de clasificación como de regresión.
- Al utilizar múltiples árboles se reduce considerablemente el riesgo de sobreajuste.
- Se mantiene estable ante nuevas muestras, ya que el promedio de las votaciones de cientos de árboles prevalece.

3. Desventajas

- En algunos casos particulares, Random Forest puede caer en sobreajuste.
- Es más costoso de crear y ejecutar que un único árbol de decisión.
- Puede requerir un tiempo de entrenamiento considerable.
- No funciona bien con datasets pequeños.
- Es muy difícil interpretar el comportamiento de los cientos de árboles que conforman el bosque, lo que puede dificultar la explicación de su comportamiento a un cliente.



2. Metodología

Para realizar esta actividad se siguieron los siguientes pasos:

1. Importación de librerías y configuración

Se importaron las librerías necesarias para la manipulación de datos, visualización y creación de modelos, además de configurar el tamaño de las gráficas:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix,
    classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from pylab import rcParams
from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import RandomOverSampler
from imblearn.combine import SMOTETomek
from imblearn.ensemble import BalancedBaggingClassifier
from collections import Counter

rcParams['figure.figsize'] = 14, 8.7  # Golden Mean
LABELS = ["Normal", "Fraud"]
```

2. Lectura y exploración de datos

Se carga el conjunto de datos `creditcard.csv` y se examina la distribución de la clase:

```
df = pd.read_csv("creditcard.csv")
print(df.shape)
print(df['Class'].value_counts(sort=True))

normal_df = df[df.Class == 0]  # registros normales
fraud_df = df[df.Class == 1]   # casos de fraude
```

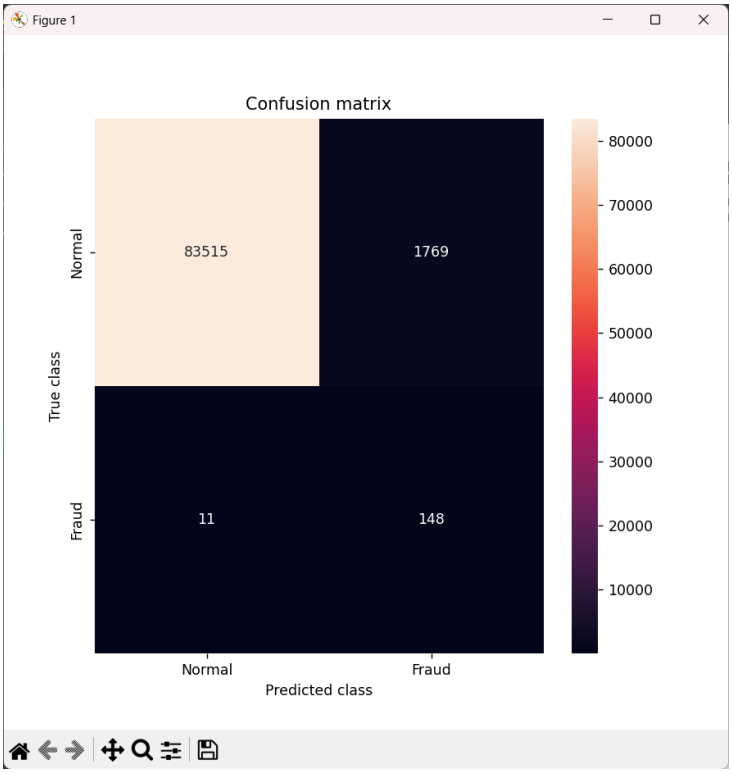
3. División de los datos y definición de funciones

Se separa la variable objetivo y las características, y se crea una función para mostrar los resultados mediante la matriz de confusión:

```
y = df['Class']
X = df.drop('Class', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    train_size=0.7)

def mostrar_resultados(y_test, pred_y):
    conf_matrix = confusion_matrix(y_test, pred_y)
    plt.figure(figsize=(8, 8))
    sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=
        LABELS, annot=True, fmt="d")
    plt.title("Confusion Matrix")
    plt.ylabel('True Class')
    plt.xlabel('Predicted Class')
    plt.show()
    print(classification_report(y_test, pred_y))
```

Visualización de los Datos No Balanceados:



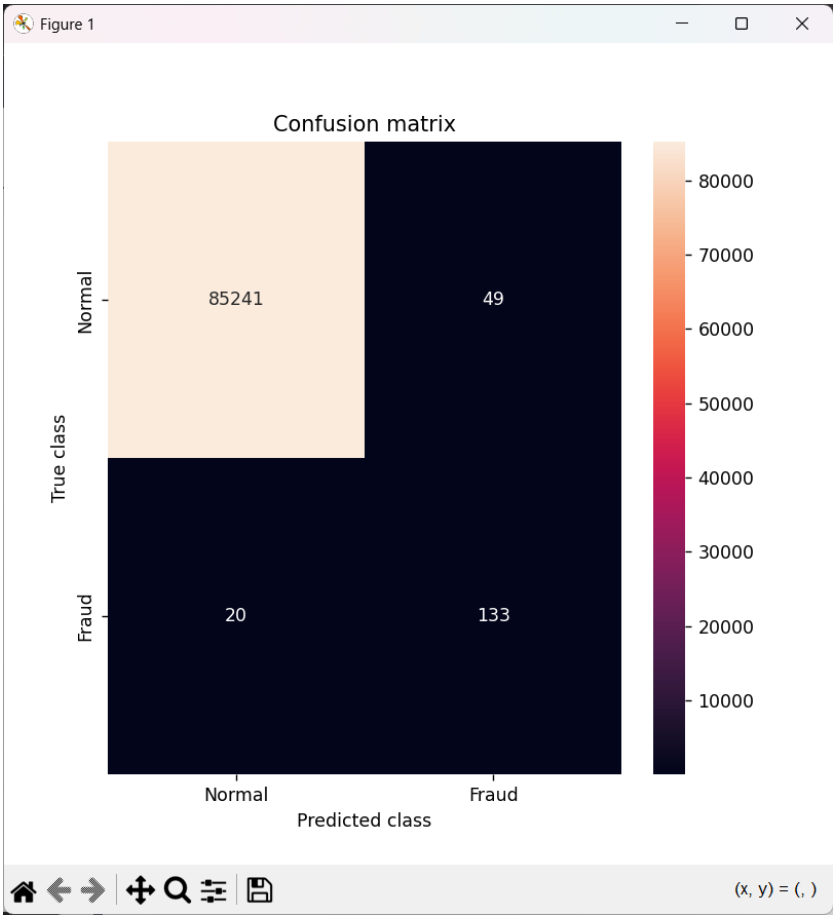
4. Modelo de Regresión Logística Balanceada

Como referencia, se entrena un modelo de Regresión Logística con pesos balanceados:

```
def run_model_balanced(X_train, X_test, y_train, y_test):
    clf = LogisticRegression(C=1.0, penalty='l2',
        random_state=1, solver="newton-cg", class_weight="
        balanced")
    clf.fit(X_train, y_train)
    return clf

model = run_model_balanced(X_train, X_test, y_train, y_test)
pred_y = model.predict(X_test)
mostrar_resultados(y_test, pred_y)
```

Visualización de los Datos Balanceados:



5. Modelo Random Forest

Se crea y entrena un modelo Random Forest con 100 árboles, ajustando parámetros para tratar el desbalanceo de clases y optimizar la profundidad:

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, class_weight
    ="balanced",
                               max_features='sqrt', verbose
                                   =2, max_depth=6,
                               oob_score=True, random_state
                                   =50, n_jobs=4)
model.fit(X_train, y_train)
pred_y = model.predict(X_test)
mostrar_resultados(y_test, pred_y)

from sklearn.metrics import roc_auc_score
roc_value = roc_auc_score(y_test, pred_y)
print(roc_value)
```

Creación del Random Forest:

```
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4
    concurrent workers.
building tree 1 of 100
building tree 2 of 100
.
.
.
building tree 99 of 100
building tree 100 of 100
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:   18.2s
    finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4
    concurrent workers.
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s
    finished
```

3. Resultados

Los resultados obtenidos en la actividad incluyen:

- **Matriz de Confusión:** Visualiza la clasificación correcta e incorrecta tanto para la clase Normal como para la de Fraude.
- **Reporte de Clasificación:** Muestra métricas como precisión, recall y f1-score.
- **ROC AUC:** La puntuación ROC AUC se utiliza para evaluar el desempeño global del modelo.

La comparación entre el modelo de Regresión Logística balanceada y el Random Forest evidencia una **mejora en la detección de fraudes**, según la métrica ROC AUC y las métricas de clasificación.

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       0.67      0.84      0.74       159

    accuracy                           1.00     85443
   macro avg       0.83      0.92      0.87     85443
weighted avg       1.00      1.00      1.00     85443
```

4. Conclusión

El modelo de **Random Forest** mostró un excelente desempeño en la detección de fraudes en transacciones con tarjeta de crédito. Se compararon los resultados obtenidos con un modelo de Regresión Logística balanceado. En metodología se incluyó la exploración y división de datos, la definición de funciones para evaluación y la optimización de parámetros en el Random Forest. Los resultados demuestran que el enfoque de Random Forest es **eficaz para tratar conjuntos de datos desbalanceados**, mejorando la capacidad de detección de casos de fraude y ofreciendo una herramienta robusta para la toma de decisiones en escenarios críticos.

```
(284807, 31)
Class
0    284315
1       492
Name: count, dtype: int64
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     85284
           1       0.08      0.93      0.14       159

    accuracy                           0.98     85443
   macro avg       0.54      0.96      0.57     85443
weighted avg       1.00      0.98      0.99     85443

[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4
   concurrent workers.
building tree 1 of 100
building tree 2 of 100
.
.
.
building tree 99 of 100
building tree 100 of 100
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:   18.2s
   finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4
   concurrent workers.
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s
   finished
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       0.67      0.84      0.74       159

    accuracy                           1.00     85443
   macro avg       0.83      0.92      0.87     85443
weighted avg       1.00      1.00      1.00     85443

0.9178461884951766
```