Random Forest en Python

31 de Marzo de 2025

1 Introducción

Random Forest es un método de aprendizaje supervisado que combina múltiples árboles de decisión entrenados con subconjuntos aleatorios de datos y características. La predicción final resulta del voto mayoritario (clasificación) o promedio (regresión) de los árboles individuales.

2 Metodología

Para realizar el ejercicio de Random Forest, se siguieron los siguientes pasos:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import confusion_matrix
from sklearn.metrics import 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

#set up graphic style in this case I am using the color scheme from xkcd.com
rcParams['figure.figsize'] = 14, 8.7 # Golden Mean
LABELS = ["Normal","Fraud"]
#col_list = ["cerulean","scarlet"]# https://xkcd.com/color/rgb/
#sns.set(style='white', font_scale=1.75, palette=sns.xkcd_palette(col_list))

%matplotlib inline
```

```
df = pd.read_csv("creditcard.csv")
df.head(n=5)
```

2.1 Análisis de los Datos

En esta sección se visualiza el desbalanceo de los datos en el dataset que se está usando. Podemos ver que existe una gran diferencia en el número de observaciones de tarjetas de crédito normales y casos de fraude.

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270

5 rows × 31 columns

```
df.shape
```

```
(284807, 31)
```

```
pd.value_counts(df['Class'], sort = True) #class comparison 0=Normal 1=Fraud
```

```
C:\Users\maxra\AppData\Local\Temp\ipykernel_9792\549919346.py:1: FutureWarning: pa
ndas.value_counts is deprecated and will be removed in a future version. Use pd.Se
ries(obj).value_counts() instead.
  pd.value_counts(df['Class'], sort = True) #class comparison 0=Normal 1=Fraud
Class
0    284315
1       492
Name: count, dtype: int64
```

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

2.2 Creación y Ajuste del Modelo

En esta sección se separan los datos en conjuntos de entrenamiento y prueba para ajustar el modelo Random Forest. No siempre es la mejor opción usar la mayor cantidad de árboles estimadores en el Random Forest, ya que la cantidad de estimadores dependerá en gran medida del dataset.

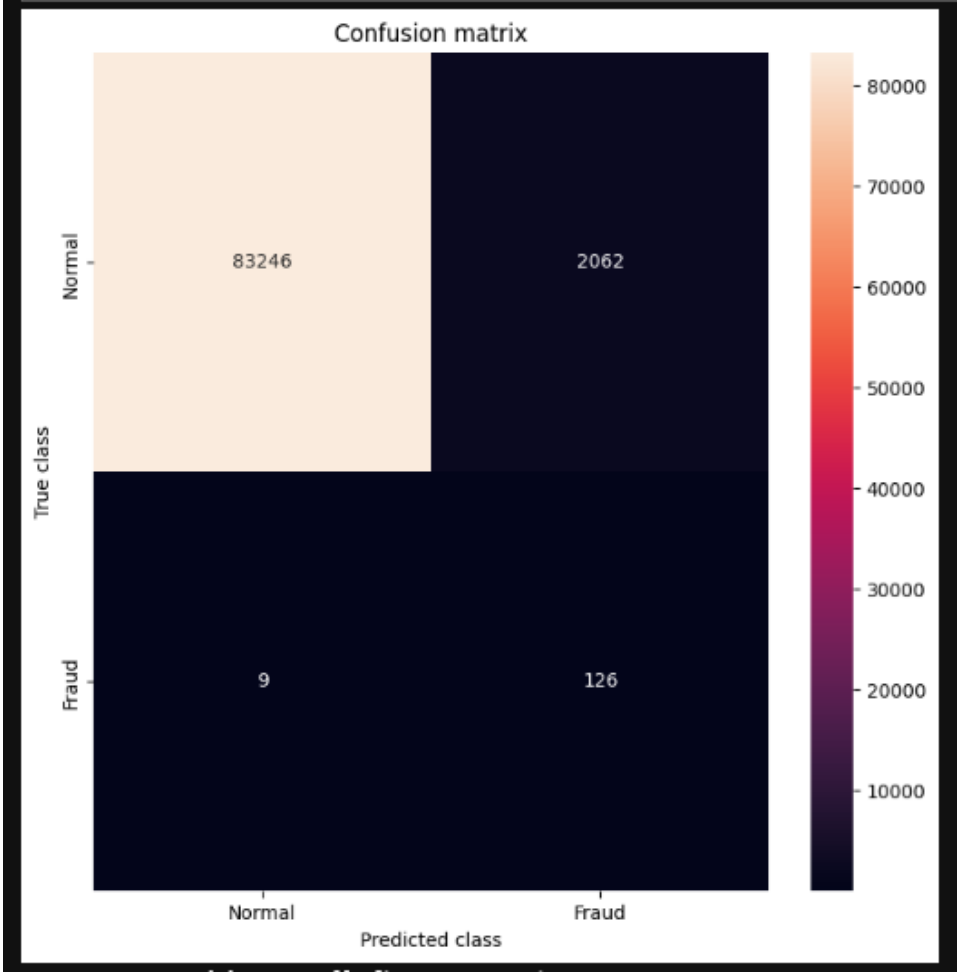
Creamos el Dataset

```
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))
```

Veamos el test set

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

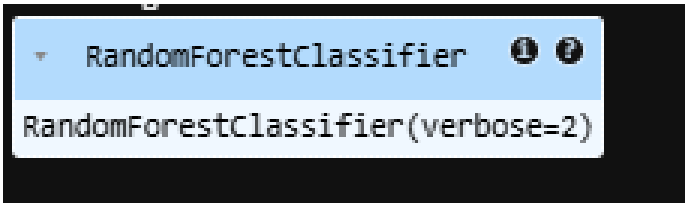


	precision	recall	f1-score	support
0	1.00	0.98	0.99	85308
1	0.06	0.93	0.11	135
accuracy			0.98	85443
macro avg	0.53	0.95	0.55	85443
weighted avg	1.00	0.98	0.99	85443

Probamos con Random Forest

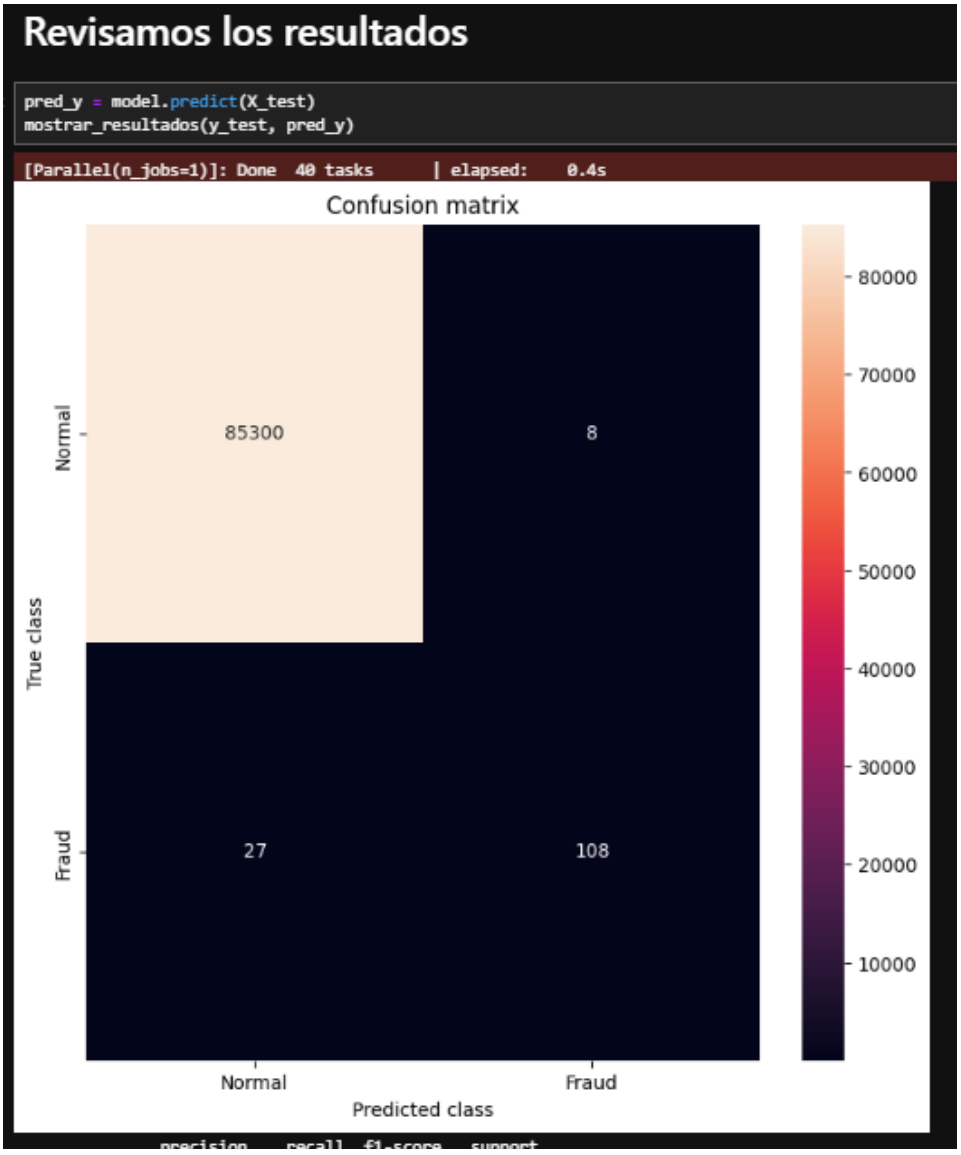
```
from sklearn.ensemble import RandomForestClassifier

# Crear el modelo con 100 arboles
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,verbose=2,
                               max_features = 'sqrt')
# entrenar!
model.fit(X_train, y_train)
```



2.3 Evaluación del Modelo

En esta sección se realizan las predicciones del modelo y se grafica la matriz de confusión. Esta matriz nos permite observar la diferencia entre la clase predecida y la clase real de cada observación del conjunto de prueba. La gran mayoría de los datos fueron asignados correctamente a su clase real, con solo 37 casos en los que el modelo se equivocó.

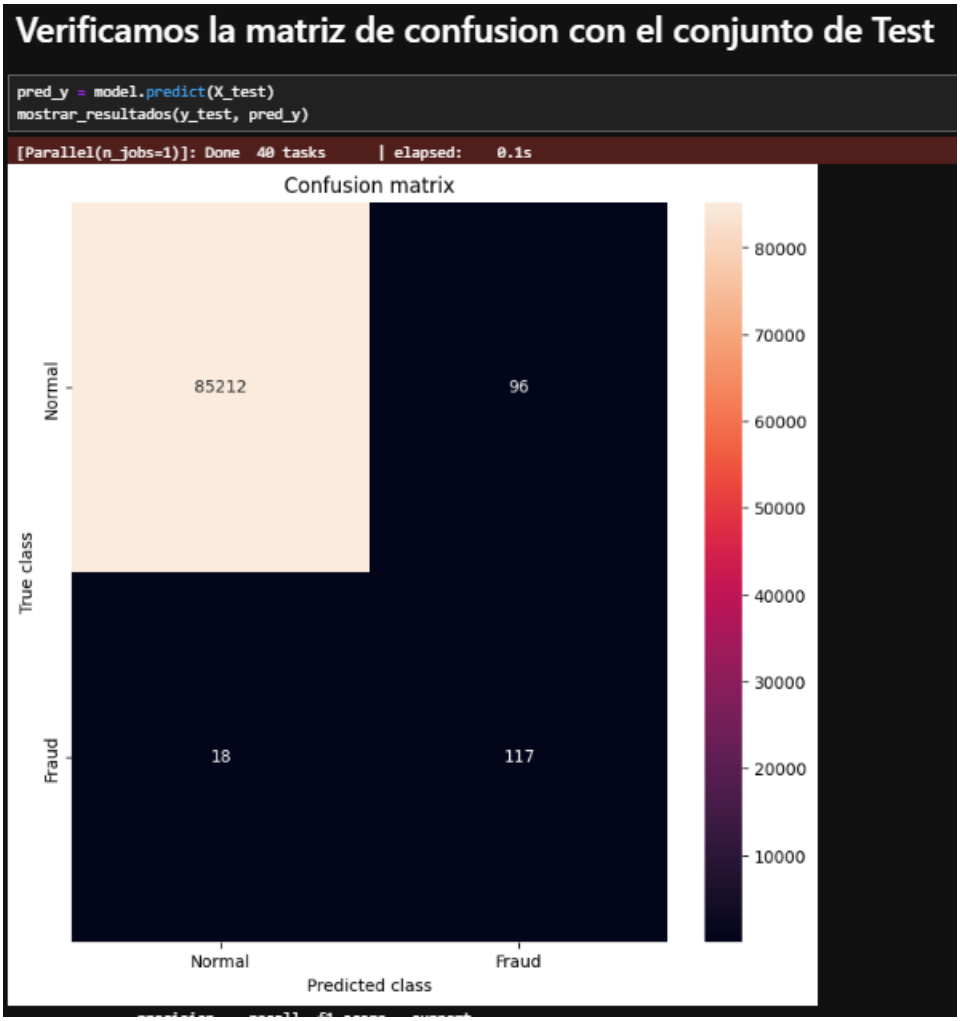


	precision	recall	f1-score	support
0	1.00	1.00	1.00	85308
1	0.93	0.80	0.86	135
accuracy			1.00	85443
macro avg	0.97	0.90	0.93	85443
weighted avg	1.00	1.00	1.00	85443

2.4 Segunda prueba de Random Forest

Creación

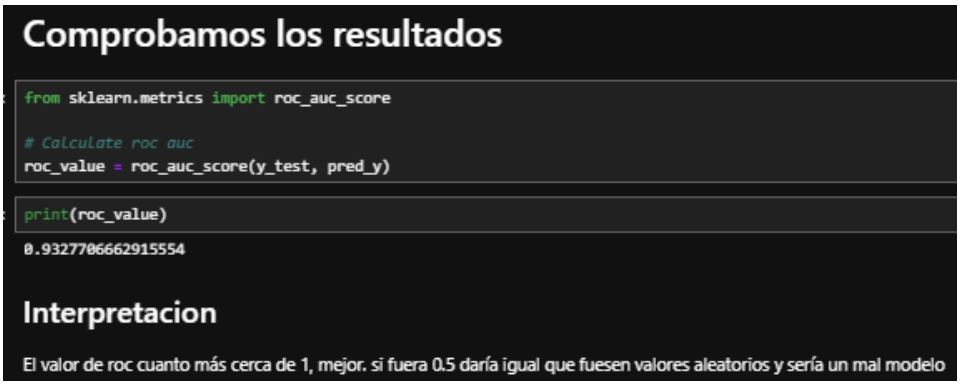
Evaluacion



	precision	recall	f1-score	support
0	1.00	1.00	1.00	85308
1	0.55	0.87	0.67	135
accuracy			1.00	85443
macro avg	0.77	0.93	0.84	85443
weighted avg	1.00	1.00	1.00	85443

3 Resultados

Otra tarea que realizamos fue comparar los resultados de este modelo con un modelo de regresión logística para analizar su precisión.



Como se puede ver, el modelo de regresión logística solo reconoció el 6% de los casos de fraude con tarjetas de crédito. Podemos confirmar que el modelo Random Forest es una mejor opción para estos datos y situaciones similares.

4 Conclusión

Los modelos de Random Forest son herramientas de aprendizaje supervisado que utilizan varios árboles de decisión que toman decisiones de forma independiente. En el caso de variables categóricas, el modelo escoge el resultado que se predijo más veces, mientras que para variables numéricas calcula el promedio de todos los resultados.

En este estudio, el modelo Random Forest tuvo un mejor desempeño en la predicción de casos de fraude en comparación con la regresión logística. Sin embargo, también se debe considerar que los modelos Random Forest tardan más en entrenarse, por lo que la decisión de utilizar Random Forest o regresión logística dependerá de las necesidades del cliente y los recursos computacionales disponibles.