Inteligencia Artificial
Act13: Random Forest

Arturo Garza Rodríguez

March 2025

1. Introducción

¿Qué es un Random Forest?

El Random Forest es un algoritmo de aprendizaje automático que utiliza un conjunto de múltiples árboles de decisión para hacer predicciones más precisas y robustas. Este método combina los resultados de varios árboles individuales, cada uno entrenado con diferentes subconjuntos de datos, para llegar a un resultado final. La técnica es muy popular debido a su flexibilidad, facilidad de uso e interpretabilidad. Además, es adecuada tanto para tareas de clasificación como de regresión. El algoritmo destaca por su estabilidad y capacidad para manejar una amplia gama de problemas en Machine Learning, ofreciendo buenas coincidencias y resistencia al sobreajuste.

2. Metodología

2.1. Requerimientos

2.1.1. Jupyter notebook y dataset

En este trabajo estarémos utilizando un entorno de una *Jupyter Notebook* para segmentar el código e ir tomando nota de los resultados con cada procedimiento.
En este caso usaremos el dataset 'creditcard.csv' de Kaggle.

2.2. Desarrollo de código

2.2.1. Imports

```python
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
```

Figura 1: Se realizan los imports de las librerías necesarias para configurar nuestro Random Forest.

2.2.2. Definición de dataset

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

Python

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137

5 rows × 31 columns

```
df.shape
```

Python

```
(284807, 31)
```

Figura 2: Se lee el dataset usando la librería pandas y se imprimen los primeros 5 registros, para ver el formato del dataset, a través de *data.head()*.
Se obtienen las dimensiones del conjuto de datos.

2.2.3. Proceso siguiente

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

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)
```

Python

```
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))
```

Python

Figura 3: Se crea un modelo que dividirá el dataset en una relación 70/30. Y se define una función para mostrar los resultados de manera más simple más adelante.

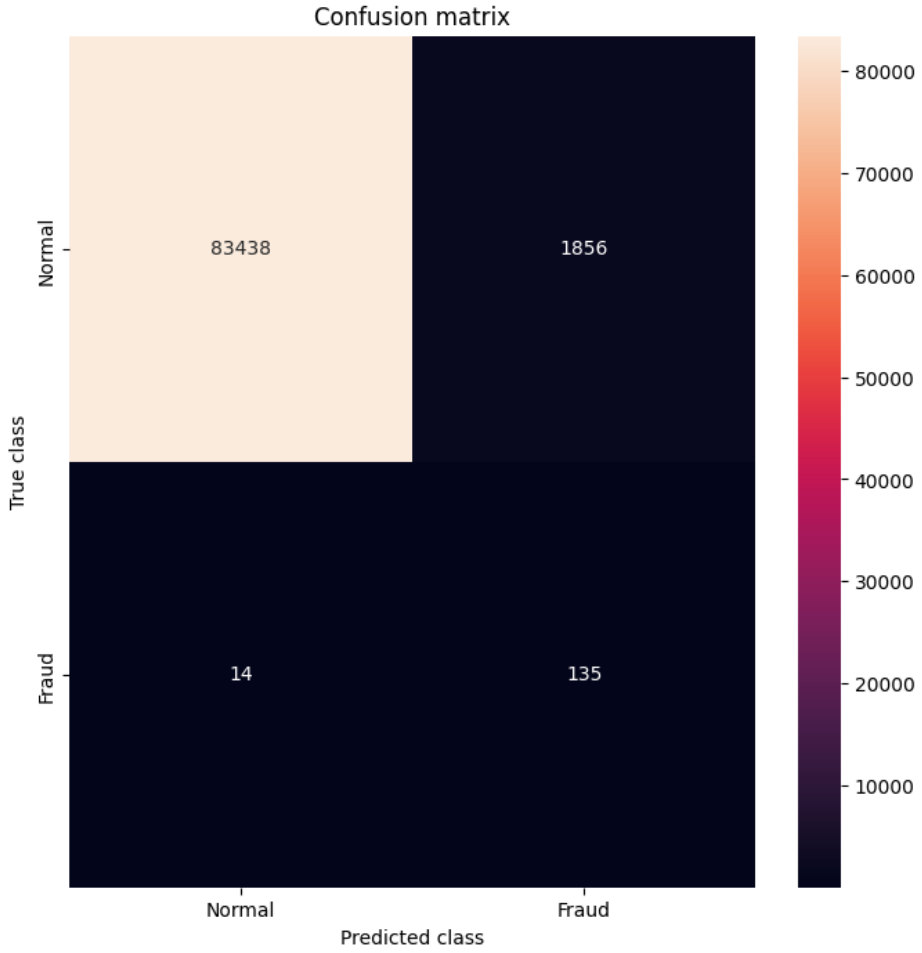
```python
def run_model_balanced(X_train, X_test, y_train, y_test):
    clf = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="newton-cg",class_weight="balanced")
    clf.fit(X_train, y_train)
    return clf

model = run_model_balanced(X_train, X_test, y_train, y_test)
```

Python

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

Python

Figura 4: Se crea un modelo de regresión logística para comparar los resultados que se obtendrán con el Random Forest más adelante y se muestra su confusion matrix.



```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     85294
           1       0.07      0.91      0.13       149

    accuracy                           0.98     85443
   macro avg       0.53      0.94      0.56     85443
weighted avg       1.00      0.98      0.99     85443
```

```python
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)
```

```
[39]
Python
```

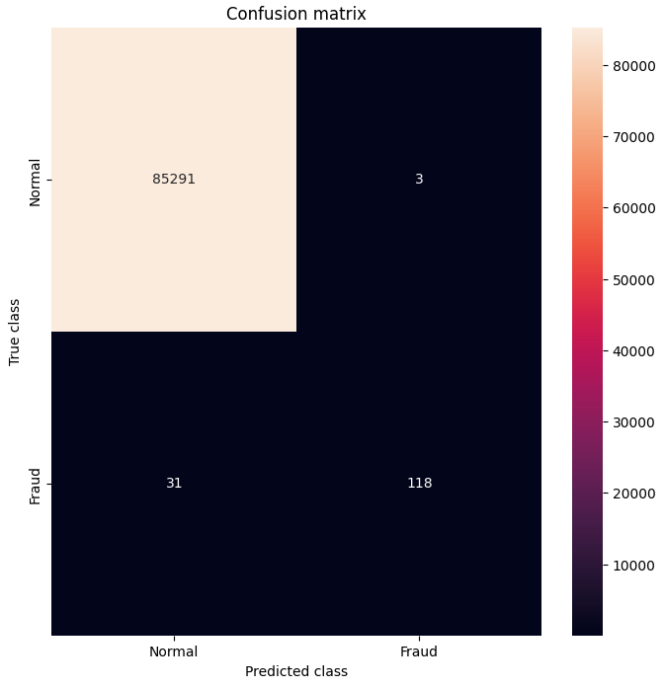
```
building tree 1 of 100
building tree 2 of 100
building tree 3 of 100
building tree 4 of 100
building tree 5 of 100
building tree 6 of 100
building tree 7 of 100
building tree 8 of 100
```

Figura 5: El texto anterior al código está asociado a la matriz anterior.
Se genera un RandomForestClassifier con n_estimators = 100.

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

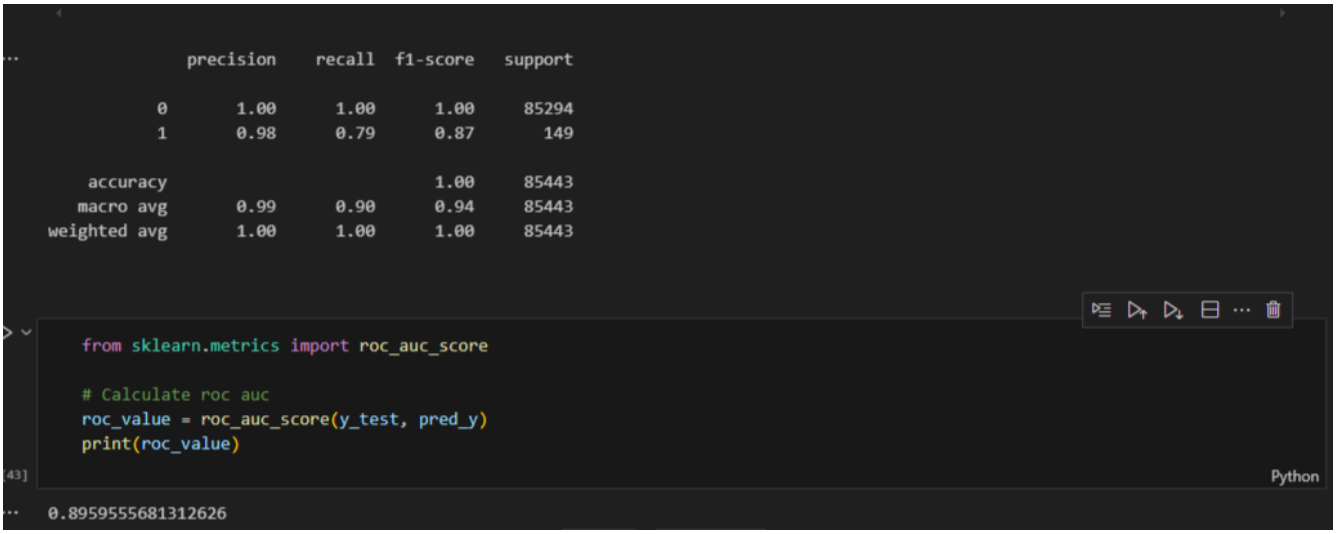
```
Python
```

Figura 6: Se muestra su confusion matrix.



3. Resultados

Dadas ambas matrices de confusión encontramos que a pesar de poseer información similar, aquella mostrada en el RandomForest es de mayor confianza.



	precision	recall	f1-score	support
0	1.00	1.00	1.00	85294
1	0.98	0.79	0.87	149
accuracy			1.00	85443
macro avg	0.99	0.90	0.94	85443
weighted avg	1.00	1.00	1.00	85443

```
from sklearn.metrics import roc_auc_score

# Calculate roc auc
roc_value = roc_auc_score(y_test, pred_y)
print(roc_value)
```

[43] Python

```
0.8959555681312626
```

Figura 7: El valor de roc cuanto más cerca de 1, mejor. si fuera 0.5 daría igual que fuesen valores aleatorios y sería un mal modelo.

4. Conclusión

En este trabajo, se implementó el algoritmo Random Forest para clasificar datos del conjunto de tarjetas de crédito. Los resultados muestran que Random Forest supera a la regresión logística en confiabilidad, con una mayor tasa de aciertos y mejor capacidad para evitar el sobreajuste. La matriz de confusión y el valor de $ROC\ AUC$ cercano a 1 demuestran la efectividad del modelo en la clasificación. En resumen, Random Forest es una técnica robusta y eficiente para problemas de clasificación.

5. Referencias

- Material de clase
- https://www.ibm.com/mx-es/think/topics/random-forest
- https://datascientest.com/es/random-forest-bosque-aleatorio-definicion-y-funcionamiento