Random Forest

Valeria Ybarra López

29, Marzo 2025

1 Introducción

Un Random Forest es un algoritmo de aprendizaje automático de uso común, que combina el resultado de múltiples árboles de decisión para llegar a un resultado único. Su facilidad de uso y flexibilidad han impulsado su adopción, ya que maneja problemas de clasificación y regresión.

Random Forest funciona así:

1. Seleccionamos k características (columnas) de las m totales, siendo $k < m$, y creamos un árbol de decisión con esas k características.
2. Creamos n árboles, variando siempre la cantidad de k características, y también podríamos variar la cantidad de muestras que pasamos a esos árboles (esto es conocido como *"bootstrap sample"*).
3. Tomamos cada uno de los n árboles y les pedimos que hagan una misma clasificación. Guardamos el resultado de cada árbol obteniendo n salidas.
4. Calculamos los votos obtenidos para cada *"clase"* seleccionada y consideraremos la más votada como la clasificación final de nuestro *"bosque"*.

2 Metodología

2.1 Carpeta Random Forest

Creamos una carpeta llamada Random Forest en donde descargaremos el dataset proporcionado por el libro para poder realizar la actividad, también crearemos en esa misma carpeta un código .py para la codificación de la actividad.

2.2 Archivo csv

Utiliazmos un dataset de kaggle con información de fraude en tarjetas de crédito. Cuenta con 284807 filas y 31 columnas de características. Nuestra salida será 0 si es un cliente "normal" o 1 si hizo uso fraudulento.

2.3 Código

Primero, importaremos las bibliotecas necesarias:

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

Cargamos Datos

Leemos el archivo csv, usamos la función .head() para ver las primeras 5 filas, y .shape para obtener el número de filas y columnas que tiene el dataframe.

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

Vemos Desbalanceo

Para visualizar la comparación de cuantos datos hay en 0=Normal y 1=Fraude, usamos la siguiente función para contar la cantidad de ocurrencias de cada valor en la columna "class" del dataframe:

```
df['Class'].value_counts(sort=True)
```

Creamos el Dataset

El código separa los registros normales (normal_df) y los registros fraude (fraud_df). Dividimos los datos en variable independiente (X) y dependiente (y). Define la función mostrar_resultados la cuál evalúa las predicciones

```
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)

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))
```

Creamos el modelo y lo entrenamos

Para el modelo utilizaremos RandomForestClassifier de Scikit-Learn.

Al ajustar el modelo debemos de contemplar los hiperparámetros, puesto que nos ayudarán a que el bosque de mejores resultados:

- **n_estimators**: será la cantidad de árboles que generaremos.
- **max_features**: la manera de seleccionar la cantidad máxima de *features* para cada árbol.
- **min_sample_leaf**: número mínimo de elementos en las hojas para permitir un nuevo *split* (división) del nodo.
- **oob_score**: es un método que emula el *cross-validation* en árboles y permite mejorar la precisión y evitar *overfitting*.
- **boostrap**: para utilizar diversos tamaños de muestras para entrenar. Si se pone en falso, utilizará siempre el dataset completo.
- **n_jobs**: si tienes múltiples *cores* en tu CPU, puedes indicar cuántos puede usar el modelo al entrenar para acelerar el entrenamiento.

```
# Crear el modelo con 100 arboles
model = RandomForestClassifier(n_estimators=100,bootstrap = True,
    verbose=2,max_features = 'sqrt')
```

3 Resultados

Cargamos datos

	Time	V1	V2	V3	...	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	...	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	...	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	...	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	...	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	...	0.219422	0.215153	69.99	0

Figure 1: Resultado de df.head(n=5), muestra las 5 filas de nuestro dataframe

```
(284807, 31)
```

Figure 2: Resultado de df.shape, tenemos 284807 filas y 31 columnas.

```
Class
0    284315
1       492
Name: count, dtype: int64
```

Figure 3: Vemos que hay un desbalance muy grande, tenemos 284315 datos clasificados como 0 (normal) y 492 como 1 (fraude).

Resultados después de entrenar el modelo

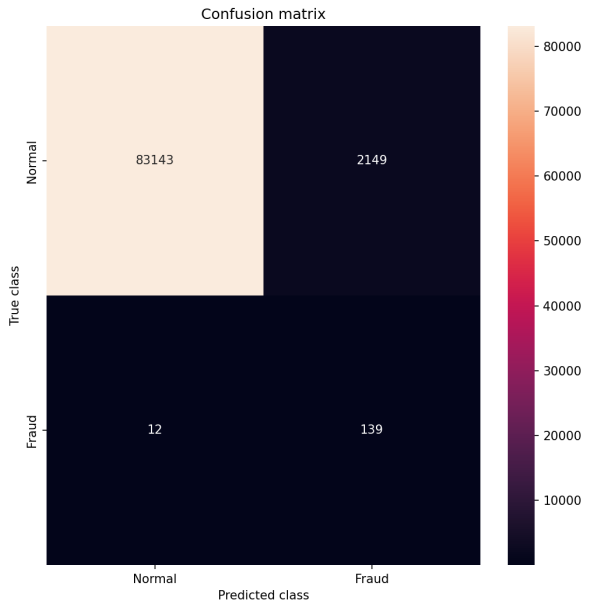
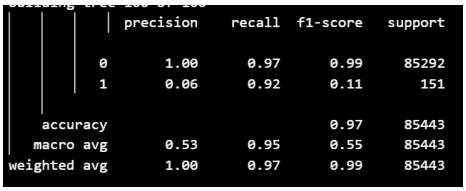


Figure 4: Vemos muy buenos resultados, clasificando con error apenas 11 + 28 muestras.



	precision	recall	f1-score	support
0	1.00	0.97	0.99	85292
1	0.06	0.92	0.11	151
accuracy			0.97	85443
macro avg	0.53	0.95	0.55	85443
weighted avg	1.00	0.97	0.99	85443

Figure 5: Podemos ver que para la clase que detecta los casos de fraude se tiene un valor de recall de 0.80, lo cual es buen indicador, y el F1-score macro avg es de 0.93.

4 Conclusión

A pesar de ser una actividad bastante corta, el algoritmo de Random Forest mostró su efectividad para tartar problemas de clasificación y regresión. En este caso, aplicamos el modelo a un dataset desbalanceado sobre fraudes con tarjetas de crédito, mostrando un buen desmepeño con métricas como un recall de 0.80 para detección de fraudes y un F1-score macro promedio de 0.93. El saber elegir nuestros hiperparámetros fueron de suma importancia ya que se obtienen resultados mas precisos.

5 Referencias

Bagnato, J. (2020). Aprende Machine Learning en Español.

IBM.¿Qué es el bosque aleatorio?. https://www.ibm.com/mx-es/think/topics/random-forest