Act13

Daniel Rojas

March 2025

1 Introducción

Un Random Forest es una serie de árboles de decisión de distinta arquitectura, los cuales en conjunto se encargan de decidir la opción correcta dado cierto input. Cada árbol "vota" por la solución correcta y la opción con más votos es la tomada como verdadera.

Para funcionar, primero se crea un árbol con un número de features igual o menor al de las features disponibles, después se le pasa una parte del dataset para su entrenamiento. Este proceso se repite con n árboles, de los cuales se selecciona la clase predecida correcta en función de los resultados de evaluar los datos de entrada en todos los árboles, tomando como correcta la clase que haya sido predecida por más árboles que las demás.

Los Random Forest pueden ser utilizados tanto para resolver problemas de clasificación como de regresión. Por otro lado, permiten evadir el overfitting facilmente, además de no necesitar modificar mucho los hiperparámetros. Sin embargo, es un método más costoso que los árboles de decisión, no funciona con conjuntos de datos pequeños y su funcionamiento es fácil de interpretar y explicar.

2 Metodología

Para realizar un Random Forest, se realizará la configuración hecha en el capítulo de Random Forest del libro Aprende Machine Learning adjuntado en las referencias.

Primero es necesario descargar el dataset de kaggle a utilizar, descomprimir el archivo, en este caso se descomprime en un google colab, cargar el dataset y, por último, crear un dataset para entrenamiento y otro para testeo. Click aquí para ir al dataset en kaggle.com

```
1 !unzip archive -d archive
```

```
#librerias a usar
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.ensemble import RandomForestClassifier

LABELS = ["Normal","Fraud"]
```

```
#Crear los datasets para entrenamiento y prueba de resultados

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)
```

Después, se define una función, tomada del propio libro, la cual sirve para crear la matriz de confusión entre los datos predecidos y los reales. La matriz de confusión muestra el número de Verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos entre dos listas de valores, por lo que es una buena forma de observar la eficacia de nuestro modelo visualmente.

```
#Muestra la matriz de confusi n entre los datos reales y los
    ↪ predecidos

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))
```

A continuación, se necesita entrenar el RandomForest:

```
# 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)
```

Por úlitmo, podemos observar la matriz de confusión y la precisión en del modelo de la siguiente forma:

```
#Ver la matriz de confusi n entre los resultados reales y los
    ↪ predichos del test dataset
pred_y = model.predict(X_test)
mostrar_resultados(y_test, pred_y)
```

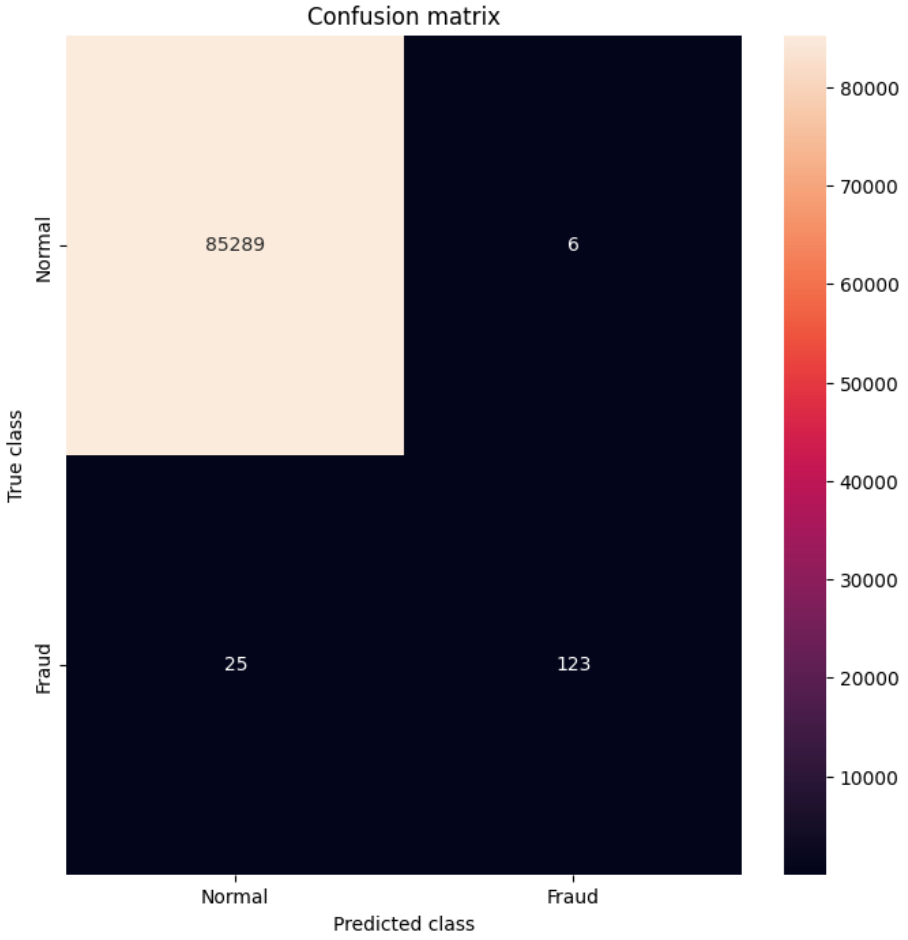

Figure 1: Matriz de confusión

```
#Calcula el roc auc, entre m s cercano a 1 m s preciso es nuestro
    ↪ modelo.
from sklearn.metrics import roc_auc_score


roc_value = roc_auc_score(y_test, pred_y)
print(roc_value)
```

Esto da como resultado un precisión del 91.55

3 Resultados

De la matriz de confusión podemos observar que claramente para datos que no son fraudes, la precisión es increiblemente grande, ya que solo se equivocó 6 veces. Sin embargo, para el caso de fraudes, la cantidad de errores es un poco más alta, pero aun así hay más predicciones correctas.

Por otro lado, la métrica auc de la curva roc muestra que en general el modelo sí es muy fiable, con un índice de precisión del 91.55

4 Conclusión

Si se cuenta con un dataset muy extenso, es recomendado utilizar el algoritmo de Random Forest, ya que puede ayudar a mitigar el overfitting y en general funciona mejor que un árbol de decisión solo. Además es muy fácil de implementar, al no necesitar modificar los hiperparámetros para obtener buenos resultados.

Por otro lado, la matriz de confusión es una buena herramienta para visualizar el rendimiento de cualquier modelo clasificador, especialmente para problemas de clasificación binarios, ya que la matriz es demasiado sencilla de leer e interpretar.

Por último, para evaluar el rendimiento de un modelo de clasificación se puede utilizar la métrica auc para una curva roc formada entre los datos predecidos y los verdaderos.

5 Referencias

Bagnato, J. I. (2020). Aprende Machine Learning.

Credit card fraud Detection. (2018, 23 marzo). Kaggle. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data