Random Forest

Jesús Roberto Dávila González

March 31, 2025

1 Introducción

Random Forest es un tipo de Ensamble en Machine Learning en donde combinaremos diversos árboles- ya veremos cómo y con qué características- y la salida de cada uno se contará como "un voto" y la opción más votada será la respuesta del Bosque Aleatorio.

Random Forest, al igual que el árbol e decisión,[1] es un modelo de aprendizaje supervisado[11] para clasificación[12] (aunque también puede usarse para problemas de regresión).

Uno de los problemas que aparecía con la creación de un árbol de decisión es que si le damos la profundidad suficiente, el árbol tiende a "memorizar" las soluciones en vez de generalizar el aprendizaje. Es decir, a padecer de overfitting[13]. La solución para evitar esto es la de crear muchos árboles y que trabajen en conjunto. Veamos cómo.

2 Metodología

```
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
```

Luego de importar las librerías que usaremos, cargamos con pandas el dataframe y vemos las primeras filas:

```
df = pd.read_csv("creditcard.csv") # read in data downloaded to the local dir
df.head(n=5)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
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	-0.110474	0.066928
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	0.101288	-0.339846
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	0.909412	-0.689281
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	-0.190321	-1.175575
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	-0.137458	0.141267

5 rows × 31 columns

124

Veamos de cuantas filas tenemos y cuantas hay de cada clase:

```
df.shape
```

```
(284807, 31)
```

Vemos desbalanceo

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

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

Vemos que son 284.807 filas y solamente 492 son la clase minoritaria con los casosde fraude. Representan el 0,17 por ciento de las muestras.

Creamos 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))
```

Ejecutamos Modelo con LogisticRegresion para poder Comparar

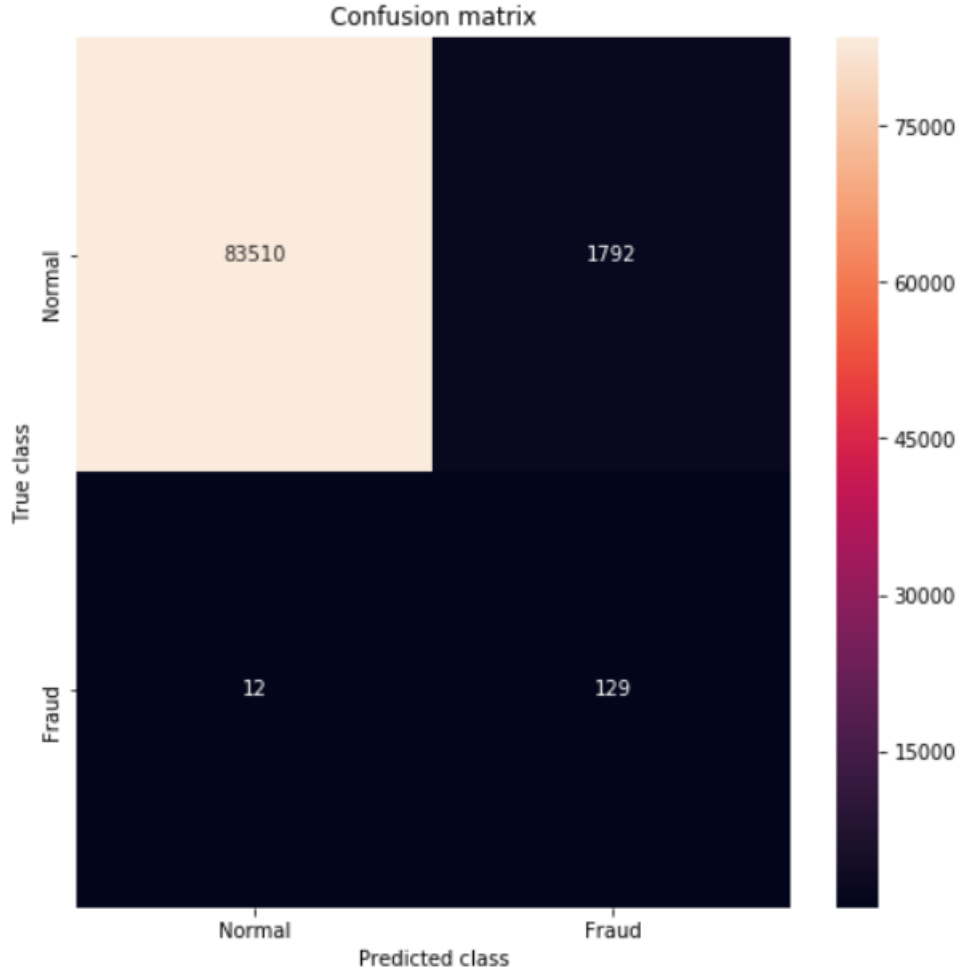
```
def run_model_balanced(X_train, X_test, y_train, y_test):
    clf = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="newton-cg",class_weight="balanced")
    clf.fit(X_train, y_train)
    return clf

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

```
/Users/jbagnato/anaconda3/envs/python36/lib/python3.6/site-packages/sklearn/utils/optimize.py:203: ConvergenceWarning: newton
-cg failed to converge. Increase the number of iterations.
  "number of iterations.", ConvergenceWarning)
```

Veamos como responde en 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	85302
1	0.07	0.91	0.13	141
accuracy			0.98	85443
macro avg	0.53	0.95	0.56	85443
weighted avg	1.00	0.98	0.99	85443

Probamos con Random Forest.

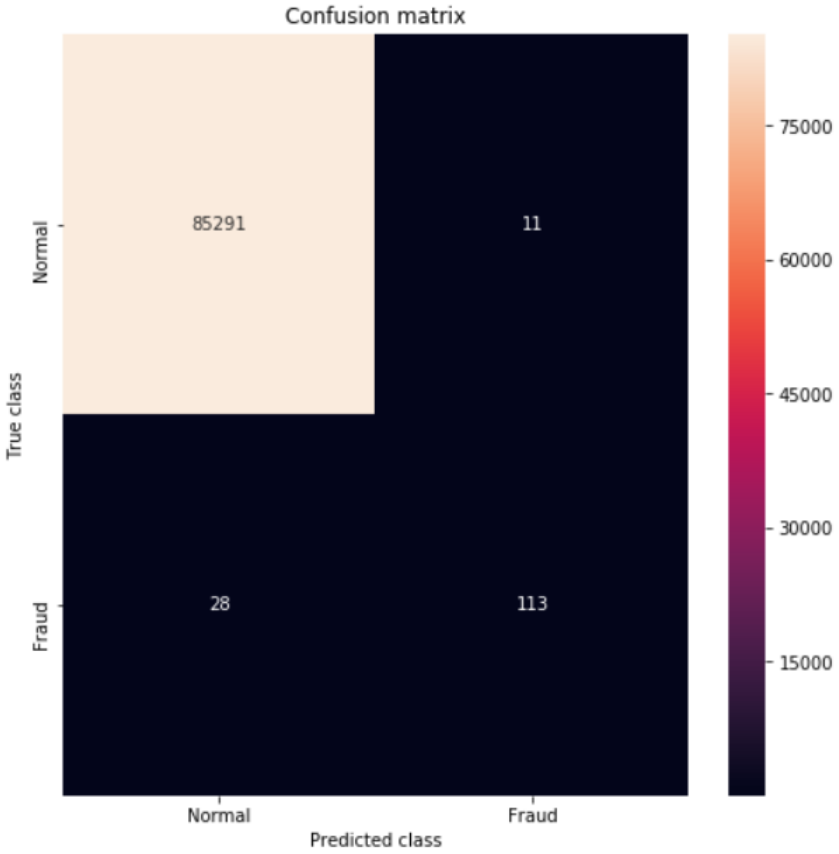
```python
from sklearn.ensemble import RandomForestClassifier

# 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)
```

Luego de unos minutos obtendremos el modelo entrenado (en mi caso 1 minuto 30 segundos)

3 Resultados

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



	precision	recall	f1-score	support
0	1.00	1.00	1.00	85302
1	0.91	0.80	0.85	141
accuracy			1.00	85443
macro avg	0.96	0.90	0.93	85443
weighted avg	1.00	1.00	1.00	85443

Aquí podemos destacar que para la clase "minoritaria", es decir la que detecta los casos de fraude tenemos un buen valor de recall (de 0.80) lo cual es un buen indicador! y el F1-score macro avg es de 0.93. Logramos construir un modelo de Bosque aleatorio que a pesar de tener un conjunto de datos de entrada muy desigual, logra buenos resultados.

4 Conclusión

Avanzando en nuestro aprendizaje sobre diversos modelos que podemos aplicar a las problemáticas que nos enfrentamos, hoy sumamos a nuestro kit de herramientas el Random Forest, vemos que es un modelo sencillo, bastante rápido y si bien perdemos la interpretabilidad maravillosa que nos brindaba 1 sólo árbol de decisión, es el precio a pagar para evitar el overfitting y para ganar un clasificador más robusto.

Los algoritmos Tree-Based-en inglés son muchos, todos parten de la idea principal de árbol de decisión y la mejoran con diferentes tipos de ensambles y técnicas. Tenemos que destacar a 2 modelos que según el caso logran superar a las mismísimas redes neuronales! son XGboost y LightGBM. Si te parecen interesantes puede que en el futuro escribamos sobre ellos.