Actividad #13: Programando Random Forest en Python

Nombre: Dayla Marely Carrizales Ortega

Matricula: 1952471

1. Introducción

Random forest es un algoritmo de aprendizaje automático supervisado que se usa para solucionar problemas de clasificación y regresión. Construye árboles de decisión a partir de diferentes muestras y toma su voto mayoritario para decidir la clasificación y el promedio en caso de regresión.

Una de las características más importantes del algoritmo de bosque aleatorio es que puede manejar un conjunto de datos que contenga variables continuas, como en el caso de la regresión, y variables categóricas, como en el caso de la clasificación. Por eso ofrece mejores resultados para problemas de clasificación.

2. Metodología

Lo primero que hicimos en esta actividad fue cargar el archivo CSV que se nos proporcionó desde la página dada en el libro. El conjunto de datos contiene varias características relacionadas con las transacciones y una columna llamada Class, que indica si la transacción es normal (0) o fraudulenta (1).

Hicimos la separación de las características de la variable dependiente (Class) y luego dividimos el conjunto de datos en subconjuntos de entrenamiento y prueba utilizando train_test_split de scikit-learn. Se entrenaron diferentes modelos de Random Forest con parámetros ajustados. En el primer modelo utilizamos un número de 100 árboles de decisión y establecimos que el modelo utiliza una submuestra aleatoria de características para cada árbol.

Después se evaluaron los modelos utilizando métricas como la matriz de confusión y el informe de clasificación. También calculamos la puntuación ROC AUC para medir la capacidad del modelo para diferenciar entre las clases (fraude y normal).

Por último, ajustamos algunos hiperparámetros, como la profundidad máxima del árbol (max_depth) y el peso de clase (class_weight), para mejorar el rendimiento del modelo.

A continuación, se mostrará el código resultante,

```
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 sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

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))

# Descargar desde https://www.kaggle.com/mlg-ulb/creditcardfraud/data

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

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

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)

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

model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,verbose=2,
                               max_features = 'sqrt')
# entrenar!
model.fit(X_train, y_train)

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

# otro modelo, variando hiperparámetros
model = RandomForestClassifier(n_estimators=100, class_weight="balanced",
                               max_features = 'sqrt', verbose=2, max_depth=6,
                               oob_score=True, random_state=50)
# a entrenar
model.fit(X_train, y_train)

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

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

3. Resultados

Los resultados obtenidos a partir de los modelos de Random Forest entrenados fueron los siguientes:

- La matriz de confusión muestra cómo el modelo ha clasificado correctamente las transacciones normales y fraudulentas, indicando cuántas veces el modelo acertó o erró en sus predicciones. Las cuales se visualizan con un mapa de calor.
- El informe de clasificación mostró varias métricas, como precisión, recall, F1-score y soporte para cada clase (fraude y normal). Estas métricas son necesarias para entender el desempeño del modelo en términos de la clasificación de transacciones fraudulentas.
- La puntuación ROC AUC calculada fue de aproximadamente 0.98, lo que indica un buen desempeño en términos de la capacidad del modelo para distinguir entre transacciones normales y fraudulentas.

4. Conclusión

En esta actividad se utilizó el algoritmo Random Forest para clasificar transacciones de tarjetas de crédito como fraudulentas o normales. El modelo fue muy efectivo para predecir el fraude en el conjunto de datos, sin embargo, el rendimiento puede mejorarse aún más ajustando los hiperparámetros del modelo y probando otras técnicas de resampling para abordar el desbalance en las clases. En conclusión, Random Forest nos demostró que es una herramienta efectiva para casos de este tipo, aunque siempre es bueno evaluar otras técnicas para obtener una mayor precisión en escenarios similares.