Actividad práctica: Análisis de precisión vs. recall en la detección de fraude

Objetivo:

Los estudiantes deben aprender a diferenciar entre **precisión** y **recall**, y entender por qué **recall** es más importante que precisión en situaciones como la **detección de fraude**, donde los **falsos negativos** pueden ser muy costosos.

Descripción del experimento:

1. **Cargar y preprocesar los datos**: Utilizaremos un conjunto de datos simulado o el conjunto de datos de **detección de fraude** (puedes usar cualquier conjunto de datos donde la clase positiva esté muy desbalanceada).
2. **Entrenar un modelo de clasificación**: Los estudiantes entrenarán un modelo (como un **Random Forest** o un **Árbol de Decisión**) para predecir si una transacción es fraudulenta o no.
3. **Evaluar el modelo con precisión y recall**: Los estudiantes evaluarán el modelo utilizando tanto **precisión** como **recall**, observando las diferencias en estas métricas cuando las clases están desbalanceadas.
4. **Analizar el impacto de un alto recall**: Los estudiantes observarán los efectos de priorizar **recall** sobre **precisión**, viendo cómo el modelo puede tener una mayor tasa de falsos positivos pero aún así capturar más fraudes.

1. Cargar y preprocesar los datos:

Usaremos un conjunto de datos simulado de **fraude** donde la clase positiva (fraude) está muy desbalanceada.

2. Entrenar el modelo:

Usamos un **Random Forest** para entrenar el modelo en el conjunto de datos.

3. Evaluar el modelo con precisión y recall:

Los estudiantes evaluarán el modelo utilizando las métricas de **precisión** y **recall**.

4. Analizar los resultados:

Los estudiantes deben observar los siguientes puntos:

- **Precisión** (accuracy): ¿Qué tan exactas son las predicciones del modelo, especialmente en las predicciones positivas?
- **Recall** (sensibilidad): ¿Cuántos de los casos realmente positivos (fraudes) fueron capturados?
- **Matriz de confusión**: Observar el número de **falsos negativos** (FN) y **falsos positivos** (FP) es crucial. Los estudiantes deben enfocarse en los **falsos negativos** porque son los que más afectan cuando se está priorizando el **recall**.

Aquí tienes el desglose del código línea por línea:

Importación de bibliotecas

```
import numpy as np
```

- **import**: Palabra clave para importar módulos en Python.
- **numpy**: Biblioteca para operaciones numéricas en matrices y arreglos.
- **as np**: Se usa un alias para acceder a la biblioteca con un nombre más corto.

```
import pandas as pd
```

- **pandas**: Biblioteca para manipulación y análisis de datos estructurados.
- **as pd**: Se define un alias para facilitar su uso en el código.

```
from sklearn.datasets import make_classification
```

- **from sklearn.datasets**: Se importa un módulo específico de la biblioteca scikit-learn.
- **make_classification**: Función que genera conjuntos de datos simulados para clasificación.

```
from sklearn.model_selection import train_test_split
```

- **from sklearn.model_selection**: Se importa el módulo para dividir datos en entrenamiento y prueba.
- **train_test_split**: Función que separa los datos en subconjuntos.

```
from sklearn.ensemble import RandomForestClassifier
```

- from sklearn.ensemble: Se importa el módulo de modelos de ensamble.
- RandomForestClassifier: Implementación del clasificador Random Forest.

```
from sklearn.metrics import precision_score, recall_score, confusion_matrix,
accuracy_score
```

- from sklearn.metrics: Se importan métricas de evaluación de modelos.
- precision_score: Mide la precisión de las predicciones positivas.
- recall_score: Mide la capacidad del modelo para detectar positivos reales.
- confusion_matrix: Matriz que muestra el rendimiento de las predicciones.
- accuracy_score: Mide la proporción total de predicciones correctas.

```
import matplotlib.pyplot as plt
```

- matplotlib.pyplot: Módulo para generar gráficos.
- as plt: Se usa un alias corto para la biblioteca.

```
import seaborn as sns
```

- seaborn: Biblioteca para visualización avanzada de datos.
- sns: Alias para acceder fácilmente a la biblioteca.

Generación del conjunto de datos

```
X, y = make_classification(
    n_samples=1000,
    n_features=20,
    n_informative=2,
    n_redundant=10,
    n_classes=2,
    weights=[0.95, 0.05],
    flip_y=0,
    random_state=42
)
```

- X, y: Se generan los datos (X = características, y = etiquetas).
- n_samples=1000: Se crean 1000 filas de datos.
- n_features=20: Cada fila tiene 20 características.

- `n_informative=2`: Solo 2 características son realmente útiles para la clasificación.
- `n_redundant=10`: Hay 10 características redundantes derivadas de las informativas.
- `n_classes=2`: Se generan dos clases (fraude y no fraude).
- `weights=[0.95, 0.05]`: La mayoría (95%) pertenece a la clase "no fraude".
- `flip_y=0`: No se añade ruido a las etiquetas.
- `random_state=42`: Se fija una semilla para obtener resultados reproducibles.

División de datos en entrenamiento y prueba

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- `X_train, y_train`: Conjunto de entrenamiento (70% de los datos).
- `X_test, y_test`: Conjunto de prueba (30% de los datos).
- `test_size=0.3`: Se reserva el 30% de los datos para evaluar el modelo.
- `random_state=42`: Se usa la misma semilla para hacer la división reproducible.

Creación y entrenamiento del modelo

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
```

- `RandomForestClassifier`: Se instancia un clasificador de tipo Random Forest.
- `n_estimators=100`: Se crean 100 árboles de decisión en el bosque.
- `random_state=42`: Se fija una semilla para reproducibilidad.

```
model.fit(X_train, y_train)
```

- `fit(X_train, y_train)`: Entrena el modelo usando los datos de entrenamiento.

Realización de predicciones

```
y_pred = model.predict(X_test)
```

- **predict(X_test)**: Genera predicciones sobre los datos de prueba.
- **y_pred**: Contiene las etiquetas predichas para X_test.

Evaluación del modelo

```
precision = precision_score(y_test, y_pred)
```

- **precision_score(y_test, y_pred)**: Calcula la precisión del modelo.
- **precision**: Proporción de verdaderos positivos sobre todos los positivos predichos.

```
recall = recall_score(y_test, y_pred)
```

- **recall_score(y_test, y_pred)**: Calcula el recall del modelo.
- **recall**: Proporción de verdaderos positivos sobre todos los positivos reales.

```
accuracy = accuracy_score(y_test, y_pred)
```

- **accuracy_score(y_test, y_pred)**: Calcula la exactitud del modelo.
- **accuracy**: Proporción total de predicciones correctas.

Impresión de resultados

```
print(f"Precisión: {precision:.2f}")
```

- **print()**: Imprime un mensaje en la consola.
- **f"Precisión: {precision:.2f}"**: Muestra la precisión con dos decimales.

```
print(f"Recall: {recall:.2f}")
```

- Muestra el recall con dos decimales.

```
print(f"Exactitud: {accuracy:.2f}")
```

- Muestra la exactitud con dos decimales.

Matriz de confusión

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

- **confusion_matrix(y_test, y_pred)**: Calcula la matriz de confusión.
- **conf_matrix**: Contiene la matriz generada.

```
print("Matriz de confusión:")
print(conf_matrix)
```

- Imprime la matriz de confusión.

Visualización de la matriz de confusión

```
plt.figure(figsize=(8, 6))
```

- **plt.figure(figsize=(8, 6))**: Crea una figura de 8x6 pulgadas.

```
sns.heatmap(
    conf_matrix,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=['No Fraude', 'Fraude'],
    yticklabels=['No Fraude', 'Fraude']
)
```

- **sns.heatmap(conf_matrix)**: Dibuja un mapa de calor con la matriz de confusión.
- **annot=True**: Muestra los valores numéricos dentro de las celdas.
- **fmt='d'**: Muestra los números como enteros.
- **cmap='Blues'**: Usa una escala de colores en tonos azules.
- **xticklabels, yticklabels**: Etiquetas en los ejes.

```
plt.title("Matriz de Confusión")
plt.xlabel("Predicción")
```

```
plt.ylabel("Real")
```

- `plt.title()`: Agrega un título al gráfico.
- `plt.xlabel()`: Etiqueta para el eje X.
- `plt.ylabel()`: Etiqueta para el eje Y.

```
plt.show()
```

- `plt.show()`: Muestra el gráfico en pantalla.