01 Actividad práctica: Evaluación realista en el Titanic: Riesgos del sobreajuste y la evaluación poco realista

Objetivo:

Los estudiantes aprenderán cómo un modelo de Machine Learning puede tener un rendimiento sobreestimado cuando el conjunto de prueba no es independiente y cómo este tipo de error puede afectar la toma de decisiones en proyectos reales.

Requisitos:

- **Python** (con bibliotecas como **pandas**, **scikit-learn**, **matplotlib**)
- **Jupyter Notebook** o entorno similar
- Conjunto de datos de Titanic disponible en [Kaggle Titanic Dataset].

Descripción de la actividad:

1. **Cargar y preprocesar los datos**: Cargar el conjunto de datos Titanic desde un archivo CSV o Kaggle, preprocesar los datos (tratamiento de valores faltantes, codificación de variables categóricas, etc.).

2. **Entrenar un modelo**: Usar un modelo de clasificación (por ejemplo, **Árbol de Decisión** o **Regresión Logística**) para predecir la supervivencia de los pasajeros basándose en características como la clase, el sexo, la edad, el puerto de embarque, etc.

3. **Evaluación del modelo**:

    - **Evaluación inicial (métrica con datos de prueba no independientes)**: Evaluar el modelo usando un conjunto de prueba que tenga algunos datos duplicados o filtrados del conjunto de entrenamiento.
    - **Evaluación con datos de prueba verdaderamente independientes**: Evaluar el modelo en un conjunto de prueba completamente independiente (datos que no se han utilizado en el entrenamiento).
4. **Observación de las métricas**: Los estudiantes compararán las métricas de rendimiento (precisión, recall, F1-score, etc.) obtenidas de ambas evaluaciones (no independiente e independiente) y observarán cómo se inflan las métricas en el primer caso.

5. **Conclusión**: Los estudiantes documentarán cómo un conjunto de prueba no independiente puede dar lugar a una evaluación sobreestimada y cómo esto puede afectar la implementación de modelos en la práctica.

Voy a explicar el código desglosando cada instrucción:

Importaciones

```
import pandas as pd
```

- `import`: Palabra clave para cargar módulos/bibliotecas
- `pandas`: Biblioteca para manipulación de datos
- `as pd`: Alias para referenciar la biblioteca más corto

```
import numpy as np
```

- `import`: Palabra clave para importación
- `numpy`: Biblioteca para cálculos numéricos
- `as np`: Alias para usar numpy

```
from sklearn.impute import SimpleImputer
```

- `from`: Palabra clave para importar específicamente
- `sklearn.impute`: Submódulo de scikit-learn
- `import SimpleImputer`: Clase específica importada para manejar valores faltantes

```
from sklearn.preprocessing import OneHotEncoder
```

- `OneHotEncoder`: Clase para transformar variables categóricas a formato binario

```
from sklearn.compose import ColumnTransformer
```

- `ColumnTransformer`: Clase para aplicar transformaciones a columnas específicas

```
from sklearn.pipeline import Pipeline
```

- `Pipeline`: Clase para encadenar múltiples transformaciones

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

- `train_test_split`: Función para dividir datos en conjuntos de entrenamiento y prueba

```
from sklearn.tree import DecisionTreeClassifier
```

- `DecisionTreeClassifier`: Algoritmo para clasificación basado en árboles de decisión

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

- `classification_report`: Función para generar métricas detalladas
- `confusion_matrix`: Función para crear matriz de verdaderos/falsos positivos/negativos
- `accuracy_score`: Función para calcular exactitud del modelo

Carga de datos

```
df = pd.read_csv('titanic.csv')
```

- `df`: Variable que almacenará el dataframe
- `=`: Operador de asignación
- `pd.read_csv()`: Método de pandas para leer archivos CSV
- `'titanic.csv'`: Nombre del archivo (string)

```
df.head()
```

- `df`: Variable dataframe
- `.head()`: Método para mostrar primeras 5 filas

Definición de características

```
numeric_features = ['Age', 'Fare']
```

- `numeric_features`: Variable lista
- `=`: Operador de asignación
- `['Age', 'Fare']`: Lista con strings de nombres de columnas numéricas

```
categorical_features = ['Sex', 'Embarked']
```

- categorical_features: Variable lista
- ['Sex', 'Embarked']: Lista con strings de nombres de columnas categóricas

Verificación de datos categóricos

```
non_empty_categorical = [col for col in categorical_features if
df[col].notna().any()]
```

- non_empty_categorical: Variable para nueva lista
- [col for col in categorical_features]: Comprensión de lista, itera sobre categorical_features
- if df[col].notna().any(): Condición que verifica que la columna tenga al menos un valor no nulo
- df[col]: Acceso a columna del dataframe
- .notna(): Método que retorna booleanos (True donde no hay nulos)
- .any(): Método que devuelve True si al menos un valor es True

Transformadores para preprocesamiento

```
numeric_transformer = SimpleImputer(strategy='mean')
```

- numeric_transformer: Variable que almacena el transformador
- SimpleImputer(): Inicialización de la clase
- strategy='mean': Parámetro que especifica usar la media para rellenar valores nulos

```
categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(drop='first'))
])
```

- categorical_transformer: Variable para el transformador de categóricas
- Pipeline([]): Constructor de pipeline que acepta lista de tuplas
- ('imputer', SimpleImputer(strategy='most_frequent')): Primer paso: nombre y objeto imputador
- strategy='most_frequent': Parámetro para usar el valor más frecuente (moda)
- ('encoder', OneHotEncoder(drop='first')): Segundo paso: nombre y codificador

- `drop='first'`: Parámetro para eliminar primera categoría y evitar colinealidad

```
preprocessor = ColumnTransformer([
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, non_empty_categorical)
])
```

- `preprocessor`: Variable para el preprocesador combinado
- `ColumnTransformer([])`: Constructor que acepta lista de tuplas
- `('num', numeric_transformer, numeric_features)`: Tupla para transformación numérica
- `('cat', categorical_transformer, non_empty_categorical)`: Tupla para transformación categórica

Selección de características

```
X = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]
```

- `X`: Variable para características/predictores
- `df[[]]`: Acceso a múltiples columnas del dataframe mediante lista
- `['Pclass', 'Sex', ...]`: Lista de nombres de columnas a seleccionar

```
y = df['Survived']
```

- `y`: Variable para la etiqueta/objetivo
- `df['Survived']`: Acceso a una columna específica del dataframe

Preprocesamiento de datos

```
X = preprocessor.fit_transform(X)
```

- `preprocessor.fit_transform(X)`: Método que aprende y transforma datos en un solo paso
- `.fit_transform()`: Método que combina ajuste y transformación
- `X`: Datos a transformar

```
X = pd.DataFrame(X)
```

- `pd.DataFrame(X)`: Constructor de pandas para crear dataframe
- `X`: Datos a convertir en dataframe

División en conjuntos de entrenamiento y prueba

```
test_size = 0.2
```

- test_size: Variable para proporción de datos de prueba
- 0.2: Valor decimal (20% para prueba)

```
random_state = 42
```

- random_state: Variable para semilla aleatoria
- 42: Valor entero para reproducibilidad

```
shuffle = True
```

- shuffle: Variable para indicar mezcla de datos
- True: Valor booleano (sí mezclar)

```
stratify = y
```

- stratify: Variable para estratificación
- y: Variable objetivo, garantiza misma proporción de clases

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state, shuffle=shuffle, stratify=stratify)
```

- X_train, X_test, y_train, y_test: Múltiples variables asignadas
- train_test_split(): Función que divide datos
- X, y: Datos a dividir
- test_size=test_size: Parámetro nombrado para tamaño de prueba
- Parámetros adicionales que usan las variables definidas anteriormente

Creación de conjunto contaminado

```
X_test_fake = pd.concat([X_test, X_train.sample(10)])
```

- X_test_fake: Variable para conjunto contaminado
- pd.concat([]): Función para concatenar dataframes
- X_train.sample(10): Método que selecciona 10 muestras aleatorias
- .sample(): Método para muestreo aleatorio

```
y_test_fake = pd.concat([y_test, y_train.sample(10)])
```

- Operación similar a la anterior para las etiquetas

Entrenamiento del modelo

```
model = DecisionTreeClassifier(random_state=42, max_depth=5, min_samples_split=5,
min_samples_leaf=2)
```

- `model`: Variable para el modelo
- `DecisionTreeClassifier()`: Constructor del modelo
- `random_state=42`: Parámetro para reproducibilidad
- `max_depth=5`: Parámetro para limitar profundidad del árbol
- `min_samples_split=5`: Parámetro para mínimo de muestras para dividir
- `min_samples_leaf=2`: Parámetro para mínimo de muestras en hojas

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

- `model.fit()`: Método para entrenar modelo
- `X_train, y_train`: Datos de entrenamiento y etiquetas

Evaluación con conjunto contaminado

```
y_pred_fake = model.predict(X_test_fake)
```

- `y_pred_fake`: Variable para predicciones
- `model.predict()`: Método para generar predicciones
- `X_test_fake`: Datos para predecir (contaminados)

```
print("Evaluación con conjunto de prueba no independiente:")
```

- `print()`: Función para mostrar texto
- `"..."`: String/cadena de texto

```
print("Accuracy:", accuracy_score(y_test_fake, y_pred_fake))
```

- `accuracy_score()`: Función de métrica de exactitud
- `y_test_fake, y_pred_fake`: Valores reales y predicciones

```
print(y_test_fake[:10])
```

- `y_test_fake[:10]`: Operación de slicing (primeros 10 elementos)

```
print("Matriz de confusión:\n", confusion_matrix(y_test_fake, y_pred_fake))
```

- `confusion_matrix()`: Función para matriz de confusión
- `\n`: Carácter de nueva línea

```
print("Reporte de clasificación:\n", classification_report(y_test_fake,
y_pred_fake))
```

- `classification_report()`: Función para informe detallado

Evaluación con conjunto limpio

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

- `y_pred`: Variable para predicciones limpias
- `X_test`: Datos de prueba independientes

```
print("Evaluación con conjunto de prueba independiente:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Matriz de confusión:\n", confusion_matrix(y_test, y_pred))
print("Reporte de clasificación:\n", classification_report(y_test, y_pred))
```

- Serie de instrucciones similares a las anteriores pero con datos independientes