Tutorial: Credit Scoring

Introducción

En este tutorial, exploraremos un proyecto completo de **credit scoring** en Python, cubriendo desde la configuración del entorno, la limpieza y exploración de los datos, hasta la construcción y evaluación de modelos. A lo largo del proceso, utilizaremos varias librerías populares como `pandas`, `numpy`, `matplotlib`, `seaborn`, y herramientas de machine learning proporcionadas por `sklearn`.

1. Configuración del entorno

Primero, importamos las librerías necesarias:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_curve, confusion_matrix
```

Descripción de librerías:

- **pandas**: Nos permite manejar tablas de datos.
- **numpy**: Facilita cálculos con números y arreglos.
- **matplotlib.pyplot**: Se usa para hacer gráficos.
- **seaborn**: Mejora y simplifica la creación de gráficos estadísticos.
- **sklearn**: Proporciona herramientas para entrenar y evaluar modelos de machine learning.

Con esto, estamos listos para trabajar con datos y visualizarlos de manera efectiva.

2. Preprocesamiento de datos

Carga de datos

Utilizamos la función `pd.read_csv()` para cargar el archivo CSV en un DataFrame de pandas:

```
data = pd.read_csv('credit_data.csv')
```

Esto nos permite trabajar fácilmente con los datos en formato tabular.

Resumen de los datos

- **`data.info()`** : Proporciona un resumen del DataFrame, mostrando el número de filas y columnas, tipos de datos y valores nulos por columna.
- **`data.describe()`** : Genera estadísticas básicas para las columnas numéricas (promedios, percentiles, valores mínimos y máximos, etc.).

Tratamiento de datos

Formateo de columnas

Convertimos la columna `default` a un formato numérico con `np.where`:

```
data['default'] = np.where(data['default'] == 'Yes', 1, 0)
```

Esto asigna el valor `1` si la condición es verdadera (`Yes`) y `0` en caso contrario.

Valores anómalos

Observamos valores anómalos como `999999999` y los reemplazamos con valores nulos:

```
data.replace(999999999, np.nan, inplace=True)
```

Manejo de valores faltantes

Cuantificamos los valores nulos con `data.isnull().sum()` y los llenamos con la mediana de cada columna:

```
data.fillna(data.median(), inplace=True)
```

Histograma de variables

Creamos un histograma para observar la distribución de cada variable:

```
num_cols = data.select_dtypes(include=['float64', 'int64']).columns

fig, axes = plt.subplots(len(num_cols), 1, figsize=(10, 20))
for i, col in enumerate(num_cols):
    data[col].hist(ax=axes[i], bins=30, edgecolor='black')
    axes[i].set_title(col)
plt.tight_layout()
plt.show()
```

Truncamiento de valores atípicos

Definimos un diccionario con límites basados en percentiles:

```
limits = {
    'income': (data['income'].quantile(0.05), data['income'].quantile(0.95)),
    'age': (data['age'].quantile(0.05), data['age'].quantile(0.95))
}

for col, (low, high) in limits.items():
    data[col] = data[col].clip(lower=low, upper=high)
```

3. Exploración de datos

Variables numéricas

Analizamos la distribución de cada variable numérica en relación al riesgo usando gráficos ECDF:

```
for col in num_cols:
    sns.ecdfplot(data=data, x=col, hue='default', palette='Set1', linewidth=2)
    plt.title(f'Distribución acumulada de {col}')
    plt.show()
```

Explicación: La función `sns.ecdfplot` genera un gráfico de distribución acumulada para observar diferencias entre grupos (`default`).

Variables cualitativas

Evaluamos la contribución al riesgo calculando el riesgo promedio global y por categoría:

```
cat_cols = data.select_dtypes(include=['object']).columns

def risk_analysis(column):
    global_risk = data['default'].mean()
    grouped = data.groupby(column)['default'].mean()
    risk_ratio = grouped / global_risk
    return pd.DataFrame({'Risk': grouped, 'Risk Ratio': risk_ratio})

for col in cat_cols:
    print(risk_analysis(col))
```

4. Construcción de modelos

Transformación de datos

Convertimos variables cualitativas a variables dummies:

```
data = pd.get_dummies(data, drop_first=True)
```

División de datos

Usamos `train_test_split` para dividir los datos en conjuntos de entrenamiento y prueba:

```
X = data.drop('default', axis=1)
y = data['default']

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

Explicación: La función `train_test_split` divide los datos en proporciones definidas (70%-30% en este caso) para garantizar que las evaluaciones del modelo sean fiables.

Entrenamiento y evaluación de modelos

El proceso general para entrenar un modelo es:

1. Crear un objeto del modelo.
2. Entrenarlo con `fit`.
3. Generar predicciones con `predict` y `predict_proba`.
4. Evaluar resultados con matrices de confusión y reportes de clasificación.

Ejemplo con Random Forest:

```
# Crear el modelo
rf_model = RandomForestClassifier(random_state=42)

# Entrenar el modelo
rf_model.fit(X_train, y_train)

# Predicciones de clasificación
y_pred = rf_model.predict(X_test)
print(classification_report(y_test, y_pred))

# Predicciones de probabilidad
y_prob = rf_model.predict_proba(X_test)[:, 1]

# Curva ROC
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label='Random Forest')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Curva ROC')
```

```
plt.legend()
plt.show()
```

5. Evaluación y selección del modelo

Una vez identificado el punto de corte óptimo basado en la curva ROC, recalculamos las predicciones y generamos reportes actualizados:

```
optimal_threshold = thresholds[np.argmax(tpr - fpr)]
final_predictions = (y_prob >= optimal_threshold).astype(int)

print(confusion_matrix(y_test, final_predictions))
print(classification_report(y_test, final_predictions))
```

Este tutorial te brinda las herramientas necesarias para desarrollar un modelo de credit scoring efectivo. Puedes adaptar cada paso a tus propios datos y necesidades específicas.