COLOMBIA POTENCIA DE LA VIDA

TIC

UNIVERSIDAD SERGIO ARBOLEDA

Análisis de Datos
Nivel Integrador

TALENTO TECH

UNIVERSIDAD SERGIO ARBOLEDA

PREPARACIÓN SEMANA 12

BOSQUES ALEATORIOS

Los bosques aleatorios son un poderoso algoritmo de aprendizaje supervisado que se utiliza tanto para clasificación como para regresión. Fueron propuestos por Leo Breiman y combinan múltiples árboles de decisión para mejorar la precisión y la generalización del modelo, son ampliamente utilizados en la práctica debido a su capacidad para producir modelos precisos y su relativa facilidad de uso. La elección de utilizarlos dependerá del contexto del problema y de las características específicas del conjunto de datos.

Conceptos Básicos:

- **Árboles de Decisión:** Los Bosques Aleatorios se construyen sobre la base de árboles de decisión. Cada árbol toma decisiones basadas en reglas de división para asignar datos a clases o predecir valores numéricos.
- **Ensamble:** La idea central de los Bosques Aleatorios es crear un "ensamble" o colección de árboles de decisión. Cada árbol se entrena de manera independiente y contribuye a la decisión final del modelo.
- **Aleatorización:** Para introducir diversidad en el ensamble, cada árbol se entrena en una muestra aleatoria del conjunto de datos y con un subconjunto aleatorio de características en cada división.

BOSQUES ALEATORIOS

Ventajas de los Bosques Aleatorios:

1. **Alta Precisión:** Los Bosques Aleatorios suelen ofrecer una alta precisión en la predicción, ya que reducen el sobreajuste inherente a los árboles de decisión individuales.
2. **Robustez:** Son resistentes al ruido y a valores atípicos en los datos debido a la naturaleza de ensamble.
3. **Manejo Automático de Características:** No requieren la selección manual de características, ya que seleccionan un subconjunto aleatorio en cada árbol.
4. **Versatilidad:** Pueden aplicarse a problemas de clasificación y regresión.
5. **Escalabilidad:** Se pueden entrenar eficientemente en conjuntos de datos grandes y en paralelo.

BOSQUES ALEATORIOS

Desventajas de los Bosques Aleatorios:

1. **Falta de Interpretación:** Aunque son precisos, la interpretación de un Bosque Aleatorio puede ser más compleja en comparación con un solo árbol de decisión.
2. **Espacio y Tiempo de Entrenamiento:** Puede haber un mayor uso de memoria y tiempo de entrenamiento, especialmente en comparación con modelos más simples.
3. **Sobreajuste en Datos Ruidosos:** Aunque son resistentes al ruido, pueden sobre ajustarse a datos extremadamente ruidosos.
4. **No siempre óptimos para Datos Lineales:** Para problemas lineales, modelos más simples pueden ser más eficaces.

IMPLEMENTACIÓN DE BOSQUES ALEATORIOS CON SCIKIT-LEARN

Para implementar Bosques Aleatorios en Python, utilizaremos la biblioteca Scikit-Learn, que proporciona implementaciones eficientes y fáciles de usar tanto para clasificación como para regresión.

El código de todos los ejemplos vistos en la clase los encontrarás en el enlace: https://acortar.link/fnQBRN

IMPLEMENTACIÓN DE BOSQUES ALEATORIOS CON SCIKIT-LEARN

Clasificación con RandomForestClassifier

```python
# Importar bibliotecas necesarias
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Cargar conjunto de datos de Iris
iris = load_iris()
X, y = iris.data, iris.target

# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar y entrenar el clasificador de Bosques Aleatorios
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = clf.predict(X_test)

# Evaluar la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo: {accuracy}')
```

IMPLEMENTACIÓN DE BOSQUES ALEATORIOS CON SCIKIT-LEARN

Regresión con RandomForestRegressor

```
# Importar bibliotecas necesarias
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Cargar conjunto de datos de diabetes
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar y entrenar el regresor de Bosques Aleatorios
regressor = RandomForestRegressor(n_estimators=100, random_state=42)
regressor.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = regressor.predict(X_test)

# Evaluar el rendimiento del modelo (Error Cuadrático Medio en este caso)
mse = mean_squared_error(y_test, y_pred)
print(f'Error Cuadrático Medio: {mse}')
```

EVALUACIÓN DE BOSQUES ALEATORIOS

La evaluación de estos modelos es crucial para comprender su rendimiento y mejorar la calidad de las predicciones. A continuación, abordaremos la evaluación de Bosques Aleatorios, tanto en problemas de clasificación como de regresión.

Métricas de Evaluación para Clasificación:

1. **Exactitud (Accuracy):**
    - Mide la proporción de predicciones correctas sobre el total de predicciones.
    - Adecuada para conjuntos de datos balanceados.

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true, y_pred)
```

EVALUACIÓN DE BOSQUES ALEATORIOS

2. **Precisión, Recuperación y Puntuación F1:** Útiles cuando hay un desequilibrio entre las clases.
   - **Precisión:** Mide la proporción de verdaderos positivos entre las predicciones positivas.
   - **Recuperación:** Mide la proporción de verdaderos positivos entre todas las instancias positivas.
   - **Puntuación F1:** Combina precisión y recuperación en una sola métrica.

```
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
```

EVALUACIÓN DE BOSQUES ALEATORIOS

3. **Matriz de Confusión:**

- Proporciona una visión detallada de los resultados de clasificación.
- Especifica el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
```

EVALUACIÓN DE BOSQUES ALEATORIOS

Métricas de Evaluación para Regresión:

1. **Error Cuadrático Medio (MSE):**
   - Mide la media de los cuadrados de las diferencias entre las predicciones y los valores reales.
   - A menores valores, mejor rendimiento.

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_true, y_pred)
```

EVALUACIÓN DE BOSQUES ALEATORIOS

Coeficiente de Determinación (R^2):

- Indica la proporción de la varianza en la variable dependiente que es predecible a partir de las variables independientes.
- Un valor cercano a 1 indica un buen ajuste.

```
from sklearn.metrics import r2_score
r2 = r2_score(y_true, y_pred)
```

EVALUACIÓN DE BOSQUES ALEATORIOS

Importancia de las Características (Feature Importance):

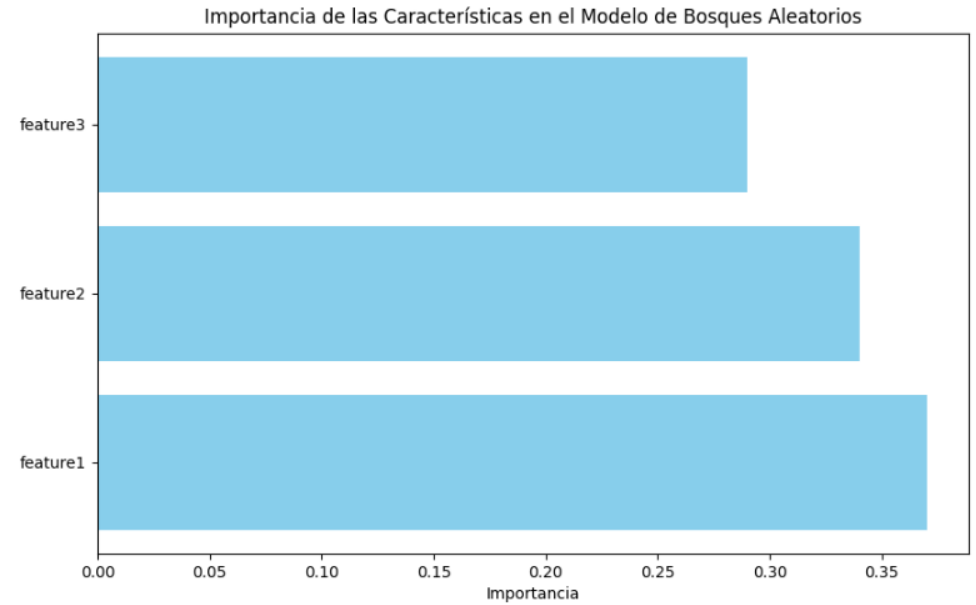
Los Bosques Aleatorios proporcionan una medida de importancia para cada característica en el conjunto de datos. Esta importancia se calcula según la contribución de cada característica al proceso de toma de decisiones del modelo. Puedes visualizar la importancia de las características de la siguiente manera:

```
importances = regressor.feature_importances_
feature_names = df.columns[:-1]  # Ajusta esto según las columnas de tu conjunto de datos
feature_importance_dict = dict(zip(feature_names, importances))

# Visualizar la importancia de las características
sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)
for feature, importance in sorted_feature_importance:
    print(f'{feature}: {importance}')
```

EVALUACIÓN DE BOSQUES ALEATORIOS

Ejemplo: Veamos cómo calcular y visualizar la importancia de las características (feature importance) con datos aleatorios (Ver código en el enlace):



APLICACIONES DE BOSQUES EN EL MUNDO REAL

Los Bosques Aleatorios son algoritmos versátiles que se pueden aplicar a una amplia variedad de problemas del mundo real en diversas industrias. Aquí hay algunos ejemplos de aplicaciones de Bosques Aleatorios:

1. **Industria de la Salud:**
    - Diagnóstico médico: Pueden utilizarse para predecir enfermedades o condiciones médicas basándose en múltiples características de los pacientes.
    - Análisis de imágenes médicas: Se aplican para segmentar y clasificar imágenes médicas, como resonancias magnéticas (RM) o tomografías computarizadas (TC).
2. **Finanzas:**
    - Evaluación de riesgos: Los Bosques Aleatorios pueden prever el riesgo crediticio y ayudar en la toma de decisiones en préstamos y finanzas personales.
    - Predicción de precios: Utilizados para predecir tendencias en los mercados financieros y en la valoración de activos.

APLICACIONES DE BOSQUES EN EL MUNDO REAL

3. **Agricultura:**
    - Rendimiento de cultivos: Pueden prever el rendimiento de cultivos en función de diversas variables, como condiciones climáticas, tipo de suelo y prácticas agrícolas.
    - Detección de plagas: Aplicados para identificar y prever la presencia de plagas en los cultivos.
4. **Comercio Minorista:**
    - Previsión de demanda: Ayudan a prever la demanda de productos en función de múltiples factores, como estacionalidad, promociones y tendencias del mercado.
    - Segmentación de clientes: Utilizados para segmentar clientes según sus comportamientos de compra y preferencias.

APLICACIONES DE BOSQUES EN EL MUNDO REAL

5. **Recursos Humanos:**
   - Selección de personal: Pueden ser utilizados para analizar currículos y predecir la idoneidad de candidatos para un trabajo específico.
   - Retención de empleados: Ayudan a identificar factores que contribuyen a la retención o rotación de empleados.
6. **Medio Ambiente:**
   - Monitoreo y conservación de la biodiversidad: Aplicados para analizar datos sobre especies y hábitats con el fin de tomar decisiones informadas sobre conservación.
   - Predicción de incendios forestales: Pueden utilizarse para prever la probabilidad de incendios forestales basándose en condiciones climáticas y otros factores.

TUNING DE HIPERPARÁMETROS PARA MEJORAR EL RENDIMIENTO

¿Qué son los Hiperparámetros?

Los hiperparámetros son configuraciones externas que no se aprenden directamente del conjunto de datos durante el entrenamiento del modelo. Son decisiones que debes tomar antes de entrenar el modelo y afectan la complejidad y el rendimiento del algoritmo. Ejemplos comunes incluyen la tasa de aprendizaje en algoritmos de aprendizaje automático, la profundidad máxima de un árbol de decisión o el número de vecinos en k-NN.

Importancia del Ajuste de Hiperparámetros:

- **Impacto en el Rendimiento del Modelo:** La elección adecuada de hiperparámetros puede mejorar significativamente el rendimiento del modelo. Unos hiperparámetros mal ajustados pueden llevar a modelos ineficaces o sobre ajustados.
- **Prevención de Sobreajuste:** Un ajuste inadecuado de los hiperparámetros puede conducir al sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos.

TUNING DE HIPERPARÁMETROS PARA MEJORAR EL RENDIMIENTO

Técnicas Comunes de Ajuste de Hiperparámetros:

1. **Búsqueda Grid (Grid Search):** Consiste en definir un conjunto de valores posibles para cada hiperparámetro y evaluar el rendimiento del modelo para todas las combinaciones posibles. Sklearn proporciona la clase **GridSearchCV** para realizar esta búsqueda.
2. **Búsqueda Aleatoria (Random Search):** Similar a la búsqueda grid, pero en lugar de evaluar todas las combinaciones posibles, selecciona aleatoriamente un conjunto de configuraciones y evalúa su rendimiento. Puede ser más eficiente en términos de tiempo.

REDES NEURONALES Y APRENDIZAJE PROFUNDO

Las redes neuronales son modelos inspirados en el funcionamiento del cerebro humano. Consisten en capas de nodos, llamados neuronas, conectadas entre sí mediante conexiones ponderadas. El aprendizaje profundo se refiere a la capacitación de redes neuronales profundas, que tienen múltiples capas intermedias (capas ocultas) entre la entrada y la salida.

Elementos Clave:

1. **Neurona Artificial:** Imita el comportamiento de una neurona biológica. Recibe entradas, aplica pesos a esas entradas, suma los resultados y aplica una función de activación.
2. **Capa:** Un conjunto de neuronas agrupadas. Las redes neuronales suelen tener capas de entrada, capas ocultas y capas de salida.
3. **Peso:** Factor de ponderación aplicado a la entrada de una neurona. Aprendido durante el entrenamiento.
4. **Función de Activación:** Define la salida de una neurona dada su entrada ponderada. Ejemplos incluyen la función sigmoide, tangente hiperbólica y ReLU (Rectified Linear Unit).

REDES NEURONALES Y APRENDIZAJE PROFUNDO

Diferencias entre CNN, RNN y Otros Tipos de Redes

1. **Redes Neuronales Convolucionales (CNN):**
   - *Uso Principal:* Para datos con estructura espacial, como imágenes.
   - *Características Clave:* Utilizan capas convolucionales para aprender patrones espaciales y jerárquicos en los datos. Efectivas en tareas de visión por computadora.
2. **Redes Neuronales Recurrentes (RNN):**
   - *Uso Principal:* Para datos secuenciales, como series temporales o procesamiento de lenguaje natural.
   - *Características Clave:* Mantienen una memoria interna que les permite trabajar con secuencias de longitud variable. Útiles en tareas donde el contexto temporal es esencial.

REDES NEURONALES Y APRENDIZAJE PROFUNDO

3. **Redes Neuronales Generativas (GAN):**
   - *Uso Principal:* Generación de datos nuevos y realistas.
   - *Características Clave:* Combinan dos redes, un generador que crea datos y un discriminador que evalúa su autenticidad. Se utilizan en la generación de imágenes, sonido y texto.
4. **Redes Neuronales Siamesas:**
   - *Uso Principal:* Comparación de similitudes entre dos entradas.
   - *Características Clave:* Comparten pesos entre dos ramas idénticas de la red. Útiles en tareas de reconocimiento de objetos, verificación de firma, etc.
5. **Redes Neuronales Feedforward (FNN):**
   - *Uso Principal:* Para tareas de clasificación y regresión.
   - *Características Clave:* La información fluye en una dirección, desde la entrada hasta la salida, sin ciclos ni retroalimentación. Capas ocultas procesan la información de manera no secuencial.

REDES NEURONALES Y APRENDIZAJE PROFUNDO

Importancia del Aprendizaje Profundo

- **Representaciones Jerárquicas:** Las capas intermedias aprenden representaciones cada vez más abstractas de los datos.
- **Capacidad de Generalización:** Las redes profundas pueden aprender automáticamente características relevantes para la tarea, evitando la necesidad de ingeniería de características manual.
- **Avances en Diversas Áreas:** Contribuciones significativas en visión por computadora, procesamiento de lenguaje natural, reconocimiento de voz y más.
- **Desafíos y Consideraciones Éticas:** A medida que los modelos se vuelven más complejos, también aumentan los desafíos éticos, como la interpretabilidad y la equidad.

IMPLEMENTACIÓN CON TENSORFLOW/KERAS

1. Construcción de Modelos Secuenciales:

En Keras, la construcción de modelos secuenciales es apropiada para una pila lineal de capas, donde la salida de una capa es la entrada de la siguiente. Aquí hay un ejemplo simple(Ver código en el enlace):

```
Epoch 1/10
3/3 [==============================] - 2s 398ms/step - loss: 0.8904 - accuracy: 0.4500 - val_loss: 1.0033 - val_accuracy: 0.4000
Epoch 2/10
3/3 [==============================] - 0s 102ms/step - loss: 0.8154 - accuracy: 0.5375 - val_loss: 0.9952 - val_accuracy: 0.4000
Epoch 3/10
3/3 [==============================] - 0s 81ms/step - loss: 0.7539 - accuracy: 0.5625 - val_loss: 0.9897 - val_accuracy: 0.3500
Epoch 4/10
3/3 [==============================] - 0s 78ms/step - loss: 0.7011 - accuracy: 0.6250 - val_loss: 0.9857 - val_accuracy: 0.4000
Epoch 5/10
3/3 [==============================] - 0s 117ms/step - loss: 0.6527 - accuracy: 0.6625 - val_loss: 0.9844 - val_accuracy: 0.3500
Epoch 6/10
3/3 [==============================] - 0s 110ms/step - loss: 0.6110 - accuracy: 0.7000 - val_loss: 0.9854 - val_accuracy: 0.3000
Epoch 7/10
3/3 [==============================] - 0s 57ms/step - loss: 0.5710 - accuracy: 0.7375 - val_loss: 0.9870 - val_accuracy: 0.3500
Epoch 8/10
3/3 [==============================] - 0s 70ms/step - loss: 0.5347 - accuracy: 0.7750 - val_loss: 0.9909 - val_accuracy: 0.3000
Epoch 9/10
3/3 [==============================] - 0s 55ms/step - loss: 0.5039 - accuracy: 0.8000 - val_loss: 0.9968 - val_accuracy: 0.3000
Epoch 10/10
3/3 [==============================] - 0s 57ms/step - loss: 0.4742 - accuracy: 0.8000 - val_loss: 1.0033 - val_accuracy: 0.2500
<keras.src.callbacks.History at 0x7da8adbc3310>
```

IMPLEMENTACIÓN CON TENSORFLOW/KERAS

2. Construcción de Modelos Funcionales:

Los modelos funcionales son más flexibles y permiten la creación de modelos con múltiples entradas o salidas. Aquí hay un ejemplo básico:

```
Epoch 1/10
3/3 [==============================] - 1s 139ms/step - loss: 0.7741 - accuracy: 0.5500 - val_loss: 0.8077 - val_accuracy: 0.5500
Epoch 2/10
3/3 [==============================] - 0s 18ms/step - loss: 0.7191 - accuracy: 0.5875 - val_loss: 0.8077 - val_accuracy: 0.5500
Epoch 3/10
3/3 [==============================] - 0s 20ms/step - loss: 0.6796 - accuracy: 0.6000 - val_loss: 0.8101 - val_accuracy: 0.5500
Epoch 4/10
3/3 [==============================] - 0s 16ms/step - loss: 0.6458 - accuracy: 0.6125 - val_loss: 0.8147 - val_accuracy: 0.5500
Epoch 5/10
3/3 [==============================] - 0s 16ms/step - loss: 0.6109 - accuracy: 0.6500 - val_loss: 0.8195 - val_accuracy: 0.5500
Epoch 6/10
3/3 [==============================] - 0s 15ms/step - loss: 0.5812 - accuracy: 0.6625 - val_loss: 0.8254 - val_accuracy: 0.5500
Epoch 7/10
3/3 [==============================] - 0s 15ms/step - loss: 0.5532 - accuracy: 0.6750 - val_loss: 0.8319 - val_accuracy: 0.5000
Epoch 8/10
3/3 [==============================] - 0s 15ms/step - loss: 0.5268 - accuracy: 0.7125 - val_loss: 0.8378 - val_accuracy: 0.5000
Epoch 9/10
3/3 [==============================] - 0s 16ms/step - loss: 0.5029 - accuracy: 0.7500 - val_loss: 0.8451 - val_accuracy: 0.5000
Epoch 10/10
3/3 [==============================] - 0s 17ms/step - loss: 0.4822 - accuracy: 0.7625 - val_loss: 0.8516 - val_accuracy: 0.5000
<keras.src.callbacks.History at 0x7da89dbdc7c0>
```

IMPLEMENTACIÓN CON TENSORFLOW/KERAS

Ejemplo práctico de clasificación de una red neuronal convolucional(CNN)

```python
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Cargar y preprocesar datos MNIST
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Construir el modelo CNN
model = keras.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compilar y entrenar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_split=0.2)
```

IMPLEMENTACIÓN CON TENSORFLOW/KERAS

Ejemplo práctico de clasificación de una red neuronal Densa

```
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

# Generar datos de ejemplo para regresión
np.random.seed(42)
X = np.random.rand(1000, 10)
y = 3 * X[:, 0] + 2 * X[:, 3] + np.random.randn(1000)

# Construir el modelo de regresión
model = keras.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10,)))
model.add(layers.Dense(1))  # Capa de salida sin función de activación para regresión

# Compilar y entrenar el modelo
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X, y, epochs=50, batch_size=32, validation_split=0.2)
```

EVALUACIÓN DE REDES NEURONALES

Métricas de Evaluación para Clasificación:

1. **Matriz de Confusión:**
   - Proporciona una visión detallada del rendimiento del modelo, mostrando la cantidad de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

```
from sklearn.metrics import confusion_matrix

y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)

conf_matrix = confusion_matrix(y_test, y_pred_binary)
print(conf_matrix)
```

EVALUACIÓN DE REDES NEURONALES

2. Precisión, Sensibilidad (Recall) y Especificidad:

- La precisión mide la proporción de predicciones positivas correctas.
- La sensibilidad (recall) mide la proporción de instancias positivas que fueron correctamente identificadas.
- La especificidad mide la proporción de instancias negativas que fueron correctamente identificadas.

```
from sklearn.metrics import precision_score, recall_score, specificity_score

precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
specificity = specificity_score(y_test, y_pred_binary)

print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Specificity: {specificity}')
```

EVALUACIÓN DE REDES NEURONALES

3. F1-Score:

La puntuación F1 es la media harmónica de precisión y sensibilidad, proporcionando un equilibrio entre ambas métricas.

```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, y_pred_binary)
print(f'F1-Score: {f1}')
```

4. Área Bajo la Curva ROC (AUC-ROC):

- Mide la capacidad del modelo para discriminar entre clases.

```
from sklearn.metrics import roc_auc_score

roc_auc = roc_auc_score(y_test, y_pred)
print(f'AUC-ROC Score: {roc_auc}')
```

EVALUACIÓN DE REDES NEURONALES

Métricas de Evaluación para Regresión:

1. **Error Absoluto Medio (MAE) y Error Cuadrático Medio (MSE):**
   - MAE mide el promedio de las diferencias absolutas entre las predicciones y los valores reales.
   - MSE mide el promedio de las diferencias al cuadrado entre las predicciones y los valores reales.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f'MAE: {mae}')
print(f'MSE: {mse}')
```

2. **Coeficiente de Determinación (R^2):**
   - Proporciona una medida de la calidad de la regresión.

```
from sklearn.metrics import r2_score

r2 = r2_score(y_test, y_pred)
print(f'R^2 Score: {r2}')
```

EVALUACIÓN DE REDES NEURONALES

Interpretación de Resultados y Diagnóstico de Problemas Comunes:

1. Análisis de Residuos:

Examinar la diferencia entre los valores predichos y los valores reales. Residuos cercanos a cero indican un buen rendimiento.

```
residuos = y_test - y_pred.flatten()

plt.scatter(y_pred, residuos)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicciones')
plt.ylabel('Residuos')
plt.title('Análisis de Residuos')
plt.show()
```

EVALUACIÓN DE REDES NEURONALES

2. **Visualización de Predicciones vs. Valores Reales:** Comparar las predicciones con los valores reales.

```
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', color='red', linewidth=2)
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Predicciones vs. Valores Reales')
plt.show()
```

¿Preguntas?