Practica5

Quiñones Mayorga Rodrigo

Parte I: Implementación de funciones sin bibliotecas para calcular medidas de desempeño

1. Accuracy (Precisión global)

```
def accuracy(y_true, y_pred):
    correct_predictions = sum([1 for true, pred in zip(y_true, y_pred) if true == pred])
    return correct_predictions / len(y_true)
```

2. Error (Tasa de error)

```
def error_rate(y_true, y_pred):
    incorrect_predictions = sum([1 for true, pred in zip(y_true, y_pred) if true != pred])
    return incorrect_predictions / len(y_true)
```

3. Matriz de confusión y medidas asociadas

Primero, se construye la **matriz de confusión** para dos clases: Positivo (1) y Negativo (0).

```
def confusion_matrix(y_true, y_pred):
    TP = sum([1 for true, pred in zip(y_true, y_pred) if true == 1 and pred == 1])
    TN = sum([1 for true, pred in zip(y_true, y_pred) if true == 0 and pred == 0])
    FP = sum([1 for true, pred in zip(y_true, y_pred) if true == 0 and pred == 1])
    FN = sum([1 for true, pred in zip(y_true, y_pred) if true == 1 and pred == 0])
    return TP, TN, FP, FN
```

A partir de esta matriz, calculamos las siguientes medidas:

- **a. Precision** (Precisión)

```
def precision(TP, FP):
    return TP / (TP + FP) if (TP + FP) > 0 else 0
```

- **b. Recall** (Sensibilidad o Recall)

```
def recall(TP, FN):
    return TP / (TP + FN) if (TP + FN) > 0 else 0
```

- **c. Positive Predictive Value (PPV)** (Valor predictivo positivo)

Este valor es el mismo que **Precision**.

- **d. True Positive Rate (TPR)** (Tasa de verdaderos positivos)

Este valor es el mismo que **Recall**.

- **e. True Negative Rate (TNR)** (Tasa de verdaderos negativos)

```
def true_negative_rate(TN, FP):
    return TN / (TN + FP) if (TN + FP) > 0 else 0
```

- **f. False Positive Rate (FPR)** (Tasa de falsos positivos)

```
def false_positive_rate(FP, TN):
    return FP / (FP + TN) if (FP + TN) > 0 else 0
```

- **g. False Negative Rate (FNR)** (Tasa de falsos negativos)

```
def false_negative_rate(FN, TP):
    return FN / (FN + TP) if (FN + TP) > 0 else 0
```

- **h. F1-Score** (Combinación de Precision y Recall)

```
def f1_score(precision, recall):
    return 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
```

Ejemplo de uso:

```
y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]  # Valores verdaderos
y_pred = [1, 0, 1, 0, 0, 1, 0, 1, 1, 0]  # Predicciones del modelo

# Obtener la matriz de confusión
TP, TN, FP, FN = confusion_matrix(y_true, y_pred)

# Calcular las métricas
precision_value = precision(TP, FP)
recall_value = recall(TP, FN)
tnr_value = true_negative_rate(TN, FP)
fpr_value = false_positive_rate(FP, TN)
fnr_value = false_negative_rate(FN, TP)
f1 = f1_score(precision_value, recall_value)
```

```
# Imprimir resultados
print(f"Precision: {precision_value}")
print(f"Recall: {recall_value}")
print(f"True Negative Rate: {tnr_value}")
print(f"False Positive Rate: {fpr_value}")
print(f"False Negative Rate: {fnr_value}")
print(f"F1-Score: {f1}")
```

Parte II: Implementación con bibliotecas de Python

Para esta parte, investigamos si las medidas anteriores están disponibles en bibliotecas de Python como **scikit-learn**.

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Calcular la matriz de confusión
cm = confusion_matrix(y_true, y_pred)
TP = cm[1, 1]
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]

# Calcular métricas usando scikit-learn
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

# Mostrar los resultados
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
```

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

print(f"F1-Score: {f1}")
```

Conclusion

- **Accuracy**, **Precision**, **Recall**, **F1-Score**, y la **matriz de confusión** están disponibles en la biblioteca **scikit-learn**.
- Esto simplifica la implementación de las métricas, ahorrando tiempo en la construcción manual de las funciones.