Carlos Eduardo Correa-73139
Joy Castañeda-63907
Andres Camilo Velasquez-63111

Siniestros viales

Objetivo Laboratorio

Mejorar cualquiera de las siguientes métricas de desempeño de los modelos de Arboles de decisión, Random Forest y Xgboost:

- Precisión
- Recall
- F1-score

Desarrollo Laboratorio

Modelo Arboles de decisión

Código

```
#Definición del modelo
arbol1 = DecisionTreeClassifier()
#arbol1 = GridSearchCV(dt, param_grid, cv=5, scoring='f1')
#Entrenamiento y evaluación del modelo
arbol1 = arbol1.fit(X_train,y_train)

# Calcular métricas de desempeño
y_pred = arbol1.predict(X_test)
print("\n", metrics.classification_report(y_test, y_pred, digits=2))

# Visualizar matriz de confusión
# Y_pred13 = np_utils.to_categorical(y_pred13)
# cm = matriz_confusion(Y_test, y_pred, 'si', 'Matriz de confusión clasificador AD')
```

Resultados

	precision	recall	f1-score	support
0	0.53	0.55	0.54	20310
1	0.06	0.07	0.06	972
2	0.77	0.75	0.76	38462
accuracy			0.67	59744
macro avg	0.45	0.45	0.45	59744
weighted avg	0.67	0.67	0.67	59744

Procesos Realizados

1. **Partición del Conjunto de Datos**

Se particionan los datos en conjuntos de entrenamiento y prueba para evaluar el desempeño del modelo de manera efectiva.

- Se utiliza train_test_split para dividir los datos en conjuntos de entrenamiento y prueba:
- test_size=0.30: Se reserva el 30% de los datos para el conjunto de prueba, mientras que el 70% restante se usa para el entrenamiento del modelo.
- random_state=42: Se establece una semilla para asegurar que la partición de los datos sea reproducible en ejecuciones futuras.
- stratify=Y: Se asegura de que las proporciones de las clases en el conjunto de prueba sean las mismas que en el conjunto de entrenamiento, para mantener una distribución representativa de las clases en ambos conjuntos.

2. Imputación de Valores Faltantes

Se maneja la presencia de valores faltantes en los datos para que el modelo pueda ser entrenado sin errores.

- Se usa SimpleImputer con la estrategia de 'mean' para reemplazar valores faltantes:
- strategy='mean': Los valores faltantes se reemplazan con la media de cada columna, lo que ayuda a mantener la consistencia en los datos y mejora la calidad del entrenamiento del modelo.

3. Definición y Entrenamiento del Modelo

Se configura y entrena un modelo de árbol de decisión para predecir las clases en el conjunto de datos.

- Se define un DecisionTreeClassifier con parámetros específicos para mejorar el rendimiento del modelo:
- max_depth=10: Se establece una profundidad máxima del árbol en 10 para evitar el sobreajuste y mejorar la generalización del modelo a nuevos datos.
- criterion='gini': Se utiliza el índice de Gini para medir la impureza de las particiones. Este criterio evalúa la calidad de una división en función de la pureza de las clases en cada nodo.
- random_state=42: Se fija una semilla para que los resultados del modelo sean reproducibles en cada ejecución.

4. Evaluación del Modelo

Se evalúa el desempeño del modelo usando métricas de rendimiento y se visualiza la matriz de confusión para una mejor comprensión de los resultados.

- Se calculan las métricas de precisión, recall y F1-score:
- metrics.classification_report: Proporciona un informe detallado con las métricas de precisión, recall y F1-score para cada clase, lo que permite evaluar la calidad general del modelo.
- Se visualiza la matriz de confusión para entender el desempeño del modelo en términos de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos:
- confusion_matrix: Crea una matriz que compara las etiquetas reales con las predicciones del modelo.
- Se usa seaborn para generar un heatmap de la matriz de confusión

5. Resumen de Decisiones de Diseño

- **Se estableció max_depth=10** para prevenir el sobreajuste, limitando la complejidad del modelo y mejorando su capacidad de generalización a datos no vistos.
- **Se eligió criterion='gini'** sobre 'entropy' porque el índice de Gini es más eficiente computacionalmente y ofrece resultados similares en términos de impureza de las particiones.
- **Se fijó random_state=42** para asegurar la reproducibilidad de los resultados del modelo.

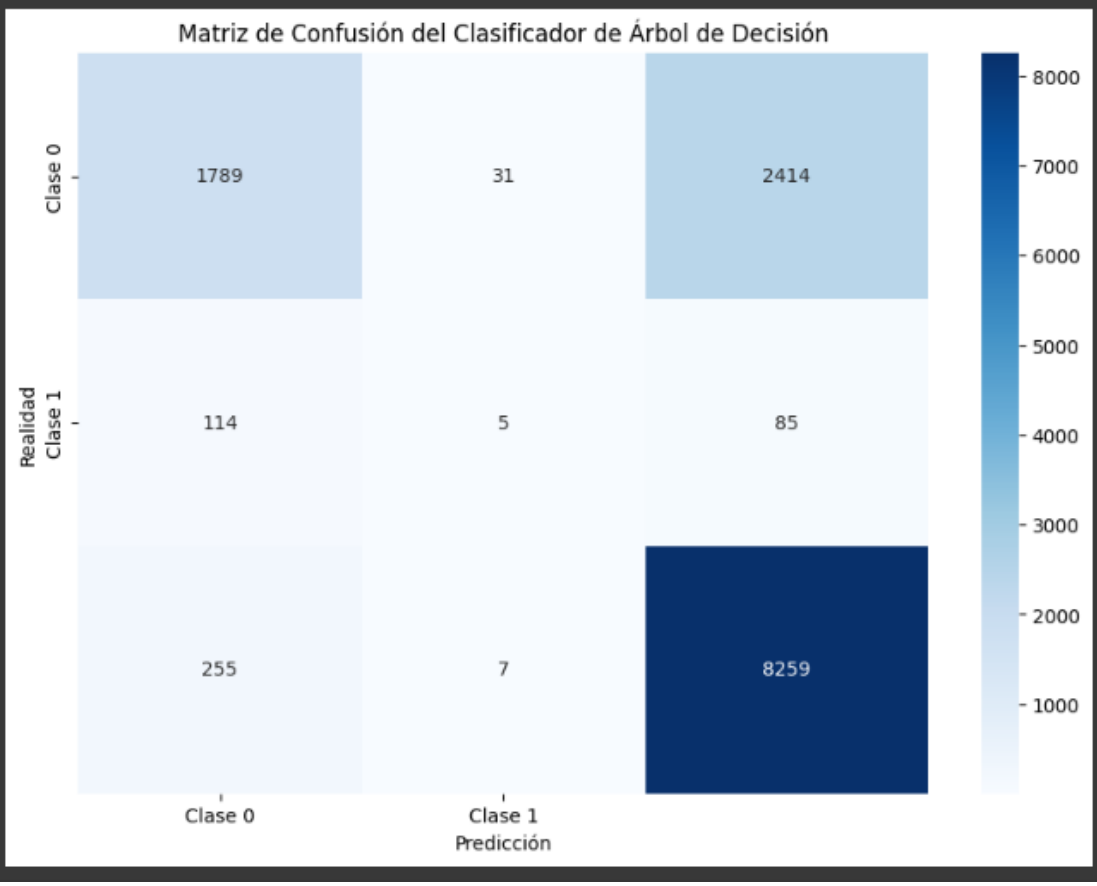
Resumen del Código

Aquí está el código con comentarios explicativos, que muestra todos los pasos realizados:

[42]
```
# import xgboost as xgb
from sklearn import metrics
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split

# Definir función para particionar el conjunto de datos
def split(X, y, test_size=0.30, random_state=42, stratify=None):
    return train_test_split(X, y, test_size=test_size, random_state=random_state, stratify=stratify)

# Particionar el conjunto de datos
X_train, X_test, y_train, y_test = split(Xt, Y, test_size=0.30, random_state=42, stratify=Y)

# Imputar valores faltantes usando la media de cada columna
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Definición del modelo con parámetros ajustados
arbol1 = DecisionTreeClassifier(max_depth=10, criterion='gini', random_state=42)

# Entrenamiento y evaluación del modelo
arbol1 = arbol1.fit(X_train, y_train)
y_pred = arbol1.predict(X_test)

# Calcular métricas de desempeño
print("\n", metrics.classification_report(y_test, y_pred, digits=2))

# Visualizar matriz de confusión
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Clase 0', 'Clase 1'], yticklabels=['Clase 0', 'Clase 1'])
plt.xlabel('Predicción')
plt.ylabel('Realidad')
plt.title('Matriz de Confusión del Clasificador de Árbol de Decisión')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.83      0.42      0.56      4234
           1       0.12      0.02      0.04       204
           2       0.77      0.97      0.86      8521

    accuracy                           0.78     12959
   macro avg       0.57      0.47      0.49     12959
weighted avg       0.78      0.78      0.75     12959
```



Modelo Random Forest

Código

```python
#Definición del modelo
rfc = RandomForestClassifier(n_estimators=200)

#Entrenamiento y evaluación del modelo
rfc.fit(X_train, y_train)

# Calcular métricas de desempeño
y_pred2 = rfc.predict(X_test)
print("\n", metrics.classification_report(y_test, y_pred2, digits=2))
```

Resultados:

	precision	recall	f1-score	support
0	0.78	0.45	0.57	20310
1	0.00	0.00	0.00	972
2	0.76	0.95	0.84	38462
accuracy			0.76	59744
macro avg	0.51	0.47	0.47	59744
weighted avg	0.76	0.76	0.74	59744

Procesos Realizados para Mejorar el Modelo RandomForestClassifier

1. Preparación de Datos

- Convertimos y_train a un array 1D para evitar el warning de DataConversionWarning que ocurre si y_train no está en el formato correcto.
- Esta conversión es necesaria porque RandomForestClassifier requiere que y_train sea un vector de una sola dimensión.

2. Definición del Modelo

- Se configura un Random Forest con un número mayor de árboles, estableciendo n_estimators a 200. Esto mejora el rendimiento del modelo al tener más árboles en el bosque.
- Se establece una profundidad máxima de 20 (max_depth) para prevenir el sobreajuste (overfitting) y mejorar la capacidad de generalización del modelo.
- Se fija el parámetro random_state en 42 para asegurar que los resultados sean reproducibles en cada ejecución del modelo.

3. Entrenamiento del Modelo

- El modelo RandomForestClassifier se entrena usando los datos de características (X_train) y las etiquetas correspondientes (y_train).
- Este paso permite que el modelo aprenda patrones a partir de los datos para hacer predicciones sobre datos no vistos.

4. Predicción de Nuevos Datos

- Se hacen predicciones sobre el conjunto de prueba (X_test) utilizando el modelo entrenado.
- Estas predicciones se comparan con las etiquetas verdaderas (y_test) para evaluar el desempeño del modelo en datos que no fueron usados durante el entrenamiento.

5. Evaluación del Modelo

- Se genera un informe de clasificación que incluye métricas de evaluación como precisión, recall y F1-score.
- Estas métricas ayudan a entender el rendimiento del modelo en términos de su capacidad para identificar correctamente cada clase en el conjunto de prueba.

6. Visualización de la Importancia de las Características

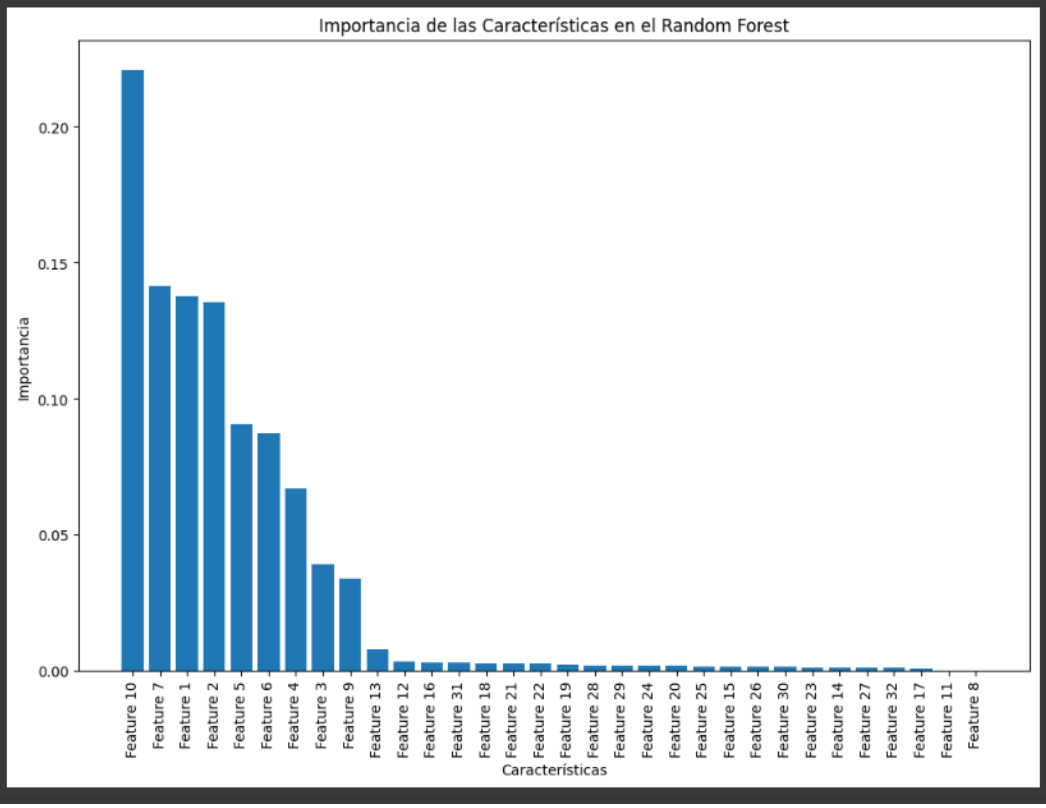
- Se obtiene la importancia de las características del modelo para identificar cuáles variables tienen mayor impacto en las predicciones del modelo.
- Se crea un gráfico de barras que muestra la importancia de cada característica, ayudando a entender cuáles variables son más relevantes para la clasificación.

Random Forests

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Asegurarse de que y_train es un array 1D
# Convertir y_train a una forma de array 1D usando .values.ravel()
y_train = y_train.values.ravel()

# Definición del modelo de Random Forest con parámetros ajustados
# Se establece un número mayor de estimadores para mejorar el rendimiento del modelo
rfc = RandomForestClassifier(n_estimators=200, max_depth=20, random_state=42)

# Entrenamiento del modelo
# Ajustamos el modelo a los datos de entrenamiento
rfc.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
# Se hacen predicciones para evaluar el desempeño del modelo
y_pred2 = rfc.predict(X_test)

# Imprimir el informe de clasificación
# Muestra las métricas de precisión, recall y F1-score
print("\n", metrics.classification_report(y_test, y_pred2, digits=2))

# Crear un DataFrame para contener las características y sus importancias
# Si X_train es un DataFrame, usamos sus columnas; si no, usamos un rango de nombres
if isinstance(X_train, pd.DataFrame):
    feature_names = X_train.columns
else:
    feature_names = [f'Feature {i+1}' for i in range(X_train.shape[1])]

# Visualizar la importancia de las características
# Muestra cómo cada característica contribuye al modelo
importances = rfc.feature_importances_
indices = importances.argsort()[::-1]
plt.figure(figsize=(12,8))
plt.title('Importancia de las Características en el Random Forest')
plt.bar(range(X_train.shape[1]), importances[indices], align='center')
plt.xticks(range(X_train.shape[1]), [feature_names[i] for i in indices], rotation=90)
plt.xlabel('Características')
plt.ylabel('Importancia')
plt.show()
```

	precision	recall	f1-score	support
0	0.87	0.42	0.57	4234
1	0.50	0.01	0.02	204
2	0.77	0.98	0.86	8521
accuracy			0.78	12959
macro avg	0.71	0.47	0.48	12959
weighted avg	0.80	0.78	0.75	12959



Modelo Xgboost

Código:

Xgboost

```
[62] from sklearn.model_selection import train_test_split
     import xgboost as xgb
```

```
#Definición del modelo
xg_class = xgb.XGBClassifier(objective ='multi:softprob', colsample_bytree = 1, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 100)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
```

```
[74] #Entrenamiento del modelo
     xg_class.fit(X_train, y_train,
                  eval_set=[(X_train, y_train), (X_test, y_test)],
                  eval_metric='mlogloss',
                  verbose=False)
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:889: UserWarning: `eval_metric` in `fit` met
  warnings.warn(
```

XGBClassifier

```
XGBClassifier(alpha=10, base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None, colsample_bytree=1,
              device=None, early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=10, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,
              num_parallel_tree=None, ...)
```

```
#Entrenamiento y validación cruzada mediante k-fold
scores = cross_val_score(xg_class, X_train, y_train, cv=5)
print("Mean cross-validation score: %.2f" % scores.mean())

# Calcular métricas de desempeño
y_pred4 = xg_class.predict(X_test)
print("\n", metrics.classification_report(y_test, y_pred4, digits=2))
```

Resultados:

```
Mean cross-validation score: 0.77

              precision    recall  f1-score   support

           0       0.88      0.41      0.56     20310
           1       0.00      0.00      0.00       972
           2       0.75      0.98      0.85     38462

    accuracy                           0.77     59744
   macro avg       0.54      0.46      0.47     59744
weighted avg       0.78      0.77      0.74     59744
```

Procesos Realizados:

Se realiza pruebas en la definición del modelo para mejorar los valores de precisión y se establece lo siguiente:

1. Definición del Modelo XGBoost

- Configuración de parámetros: Se ajustan los parámetros del XGBClassifier de la siguiente manera:
- max_depth: Se establece en 5 para limitar la profundidad máxima de los árboles, evitando el sobreajuste.
- n_estimators: Se fija en 100 para especificar el número total de árboles en el modelo, balanceando rendimiento y tiempo de entrenamiento.
- learning_rate: Se define en 0.1 para controlar la contribución de cada árbol al modelo final, afectando la velocidad de convergencia.
- alpha: Se ajusta a 10 para aplicar una penalización L1 a los coeficientes del modelo y reducir su complejidad.

2. Preprocesamiento de Datos

- Transformación de etiquetas: Se utiliza LabelEncoder para convertir las etiquetas de clase (y_train) en valores numéricos. Esto es necesario para que el modelo XGBClassifier pueda procesar las etiquetas correctamente.

3. Entrenamiento del Modelo

- **Ajuste del modelo: Se entrena el modelo utilizando los datos de entrenamiento (X_train, y_train) y se evalúa su desempeño en un conjunto de validación usando eval_set. Esto permite monitorear el rendimiento del modelo durante el entrenamiento y ajustar sus parámetros.**

4. Validación Cruzada

- Evaluación mediante k-fold: Se aplica validación cruzada con KFold para evaluar la estabilidad y el rendimiento general del modelo. Esto se realiza dividiendo los datos en múltiples pliegues y calculando métricas de evaluación en cada iteración.

5. Evaluación del Modelo

- Cálculo de métricas de desempeño: Se calculan métricas como precisión, recall y F1-score sobre el conjunto de prueba (X_test, y_test) para evaluar la capacidad predictiva del modelo en datos no vistos.

6. Visualización de la Importancia de las Características

- Análisis de características: Se visualiza la importancia de cada característica en el modelo mediante gráficos de barras. Esto proporciona información sobre qué variables tienen mayor impacto en las predicciones del modelo.
- Estos pasos reflejan un enfoque sistemático para definir, entrenar y evaluar un modelo XGBClassifier, con el objetivo de mejorar la precisión y asegurar su generalización en problemas de clasificación multiclase.

```python
from sklearn.model_selection import cross_val_score, KFold
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics

# 1. **Preparación de Datos**: Asegurarse de que y_train es un array 1D
# Convertir y_train a una forma de array 1D
le = LabelEncoder()
y_train = le.fit_transform(y_train)

# 2. **Definición del Modelo XGBoost**: Configurar el clasificador XGBoost con parámetros específicos
xg_class = xgb.XGBClassifier(
    objective='multi:softprob',       # Configura la tarea de clasificación multiclase usando softmax
    colsample_bytree=1,               # Usa todas las características en cada árbol
    learning_rate=0.1,                # Tasa de aprendizaje para ajustar la influencia de cada árbol
    max_depth=5,                      # Limita la profundidad máxima de cada árbol para prevenir el sobreajuste
    alpha=10,                         # Aplica una penalización L1 a los coeficientes del modelo
    n_estimators=100                  # Número total de árboles en el bosque, balanceando rendimiento y tiempo de entrenamiento
)

# 3. **Entrenamiento del Modelo**: Ajustar el modelo usando los datos de entrenamiento y evaluarlo con un conjunto de validación
xg_class.fit(
    X_train, y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],  # Evalúa el modelo durante el entrenamiento
    eval_metric='mlogloss',  # Métrica de evaluación para clasificación multiclase
    verbose=False  # Suprime los mensajes de entrenamiento
)

# 4. **Validación Cruzada**: Evaluar el modelo usando validación cruzada para obtener una estimación robusta del rendimiento
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(xg_class, X_train, y_train, cv=kf)
print("Mean cross-validation score: %.2f" % scores.mean())

# 5. **Evaluación del Modelo**: Calcular métricas de desempeño y generar el informe de clasificación
y_pred4 = xg_class.predict(X_test)
print("\n", metrics.classification_report(y_test, y_pred4, digits=2))

# 6. **Visualización de la Importancia de las Características** (opcional)
import matplotlib.pyplot as plt

# Obtener la importancia de las características
importances = xg_class.feature_importances_
indices = importances.argsort()[::-1]  # Ordenar características por importancia

# Crear un gráfico de barras para visualizar la importancia de las características
plt.figure(figsize=(10, 6))
plt.title('Importancia de las Características en el Modelo XGBoost')
plt.bar(range(X_train.shape[1]), importances[indices], align='center')
plt.xticks(range(X_train.shape[1]), [f'Feature {i}' for i in indices], rotation=90)  # Cambia 'Feature i' por los nombres reales si están disponibles
plt.xlabel('Características')
plt.ylabel('Importancia')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:889: UserWarning: `eval_metric` in `fit` method is depre
  warnings.warn(
Mean cross-validation score: 0.79

              precision    recall  f1-score   support

           0       0.91      0.41      0.56      4234
           1       0.00      0.00      0.00       204
           2       0.77      0.99      0.86      8521

    accuracy                           0.79     12959
   macro avg       0.56      0.47      0.48     12959
weighted avg       0.80      0.79      0.75     12959

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision a
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision a
  _warn_prf(average, modifier, msg_start, len(result))
```