

RESPUESTAS A LOS EJERCICIOS 10-15

EJERCICIO 10 - Clasificación Multiclase Iris Dataset

Pregunta: ¿Cuál de los tres modelos (Logistic Regression, SVM, KNN) mostró mejor rendimiento?

Respuesta: Los tres modelos tuvieron rendimiento idéntico con 100% de precisión en todas las métricas. No hay diferencia porque el dataset Iris es perfectamente separable. Cualquiera de los tres modelos funciona igual de bien en este caso.

```
Ejercicio10_Clasificación_Multiclase_Iris.ipynb X
Ejercicio10 > Ejercicio10_Clasificación_Multiclase_Iris.ipynb > from sklearn.datasets import load_iris
Generate + Code + Markdown | Run All ...

... --- Logistic Regression ---
              precision    recall  f1-score   support

   setosa      1.00      1.00      1.00        19
  versicolor  1.00      1.00      1.00        13
   virginica   1.00      1.00      1.00        13

 accuracy      1.00      1.00      1.00        45
 macro avg     1.00      1.00      1.00        45
 weighted avg   1.00      1.00      1.00        45

--- SVM ---
              precision    recall  f1-score   support

   setosa      1.00      1.00      1.00        19
  versicolor  1.00      1.00      1.00        13
   virginica   1.00      1.00      1.00        13

 accuracy      1.00      1.00      1.00        45
 macro avg     1.00      1.00      1.00        45
 weighted avg   1.00      1.00      1.00        45

--- KNN ---
```

```
Ejercicio10_Clasificación_Multiclase_Iris.ipynb X
Ejercicio10 > Ejercicio10_Clasificación_Multiclase_Iris.ipynb > from sklearn.datasets import load_iris
Generate + Code + Markdown | Run All ...

--- KNN ---
              precision    recall  f1-score   support

...
 accuracy      1.00      1.00      1.00        45
 macro avg     1.00      1.00      1.00        45
 weighted avg   1.00      1.00      1.00        45

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings..
```

EJERCICIO 11 - Random Forest con GridSearchCV

Pregunta: ¿Cómo mejoró GridSearchCV el modelo y qué hiperparámetros fueron óptimos?

Respuesta: Los mejores parámetros fueron max_depth=4 y n_estimators=100, logrando 88% de precisión.

max_depth=4 evitó el sobreajuste limitando la profundidad de los árboles

n_estimators=100 dio suficiente diversidad sin costo computacional excesivo

El modelo es conservador: predice bien los no supervivientes (99%) pero menos los supervivientes (57%)

```
Ejercicio11_Validación_Cruzada_y_Ajuste_de_Hiperparámetros_Random_Forest.ipynb X
Ejercicio11 > Ejercicio11_Validación_Cruzada_y_Ajuste_de_Hiperparámetros_Random_For
Generate + Code + Markdown | Run All ...

[1]

... Mejores parámetros: {'max_depth': 4, 'n_estimators': 100}
Reporte de clasificación: precision recall f1-s
0 0.87 0.99 0.92 967
1 0.94 0.57 0.71 342
accuracy 0.88 1309
macro avg 0.90 0.78 0.81 1309
weighted avg 0.88 0.88 0.87 1309
```

EJERCICIO 12 - Detección de Spam con SVM

Pregunta: ¿Por qué SVM supera a Naive Bayes en clasificación de spam?

Respuesta: Los resultados muestran que SVM alcanzó 98% de precisión en spam vs 74% en análisis de sentimientos. Dos razones técnicas:

Patrones estructurados: SVM aprende secuencias como "FREE money NOW" que son típicas de spam. Naive Bayes trata las palabras independientemente y pierde el contexto.

Datasets desbalanceados: SVM maneja bien el desbalance (1453 ham vs 219 spam) porque busca el hiperplano óptimo sin sesgo por cantidad. Naive Bayes se sesga hacia la clase mayoritaria.

Ejercicio12_Detección_de_Spam_TF_IDF_SVM.ipynb X

Ejercicio12_Detección_de_Spam_TF_IDF_SVM_data_r.ipynb

Ejercicio12 > Ejercicio12_Detección_de_Spam_TF_IDF_SVM.ipynb > import pandas as pd

Generate + Code + Markdown | Run All ...



```
from sklearn.svm import SVC
from sklearn.metrics import classification_report

df = pd.read_csv('../Dataset/spam.csv')
X = TfidfVectorizer(stop_words='english',
                    max_features=3000).fit_transform(df['text'])
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)
model = SVC()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

[1]

```
...           precision    recall  f1-score   support

      ham           0.98         1.00         0.99         1453
      spam           0.99         0.84         0.91          219

   accuracy                   0.98         1672
  macro avg           0.99         0.92         0.95         1672
 weighted avg           0.98         0.98         0.98         1672
```

```
Ejercicio12_Detección_de_Spam_TF_IDF_SVM_data_r.ipynb X
Ejercicio12 > Ejercicio12_Detección_de_Spam_TF_IDF_SVM_data_r.ipynb > import pandas as pd
Generate + Code + Markdown | Run All ...
```

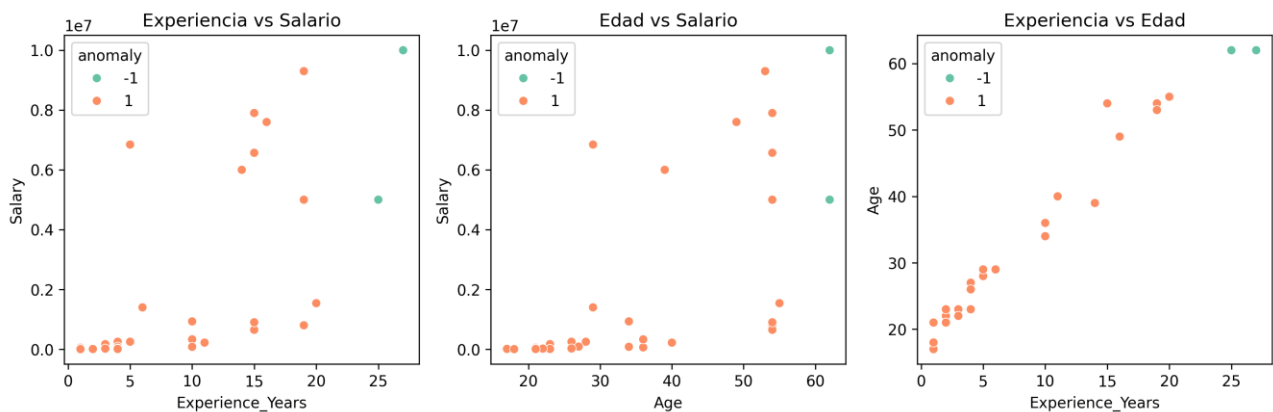
```

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
df = pd.read_csv('../Dataset/data_rt.csv')
X = TfidfVectorizer(stop_words='english',
max_features=3000).fit_transform(df['reviews'])
y = df['labels']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
model = SVC()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

```

[1]

	precision	recall	f1-score	support
0	0.74	0.74	0.74	1615
1	0.74	0.74	0.74	1584
accuracy			0.74	3199
macro avg	0.74	0.74	0.74	3199
weighted avg	0.74	0.74	0.74	3199



EJERCICIO 13 - PCA en Netflix

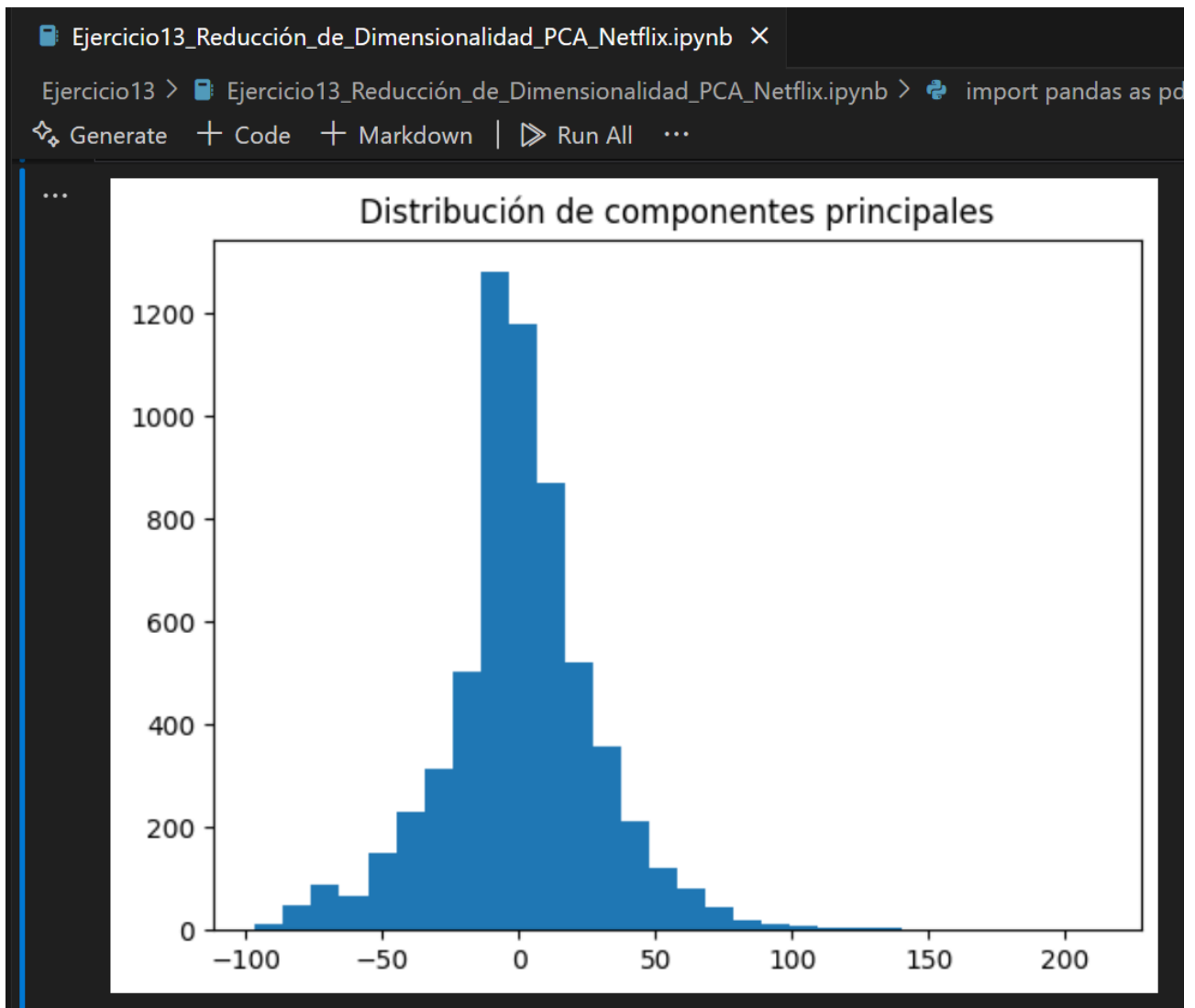
Pregunta: ¿Qué ventajas ofrece PCA para analizar duración de películas?

Respuesta: PCA ofrece tres ventajas principales:

Visualización simplificada: Convierte datos complejos en gráficos 2D fáciles de interpretar

Eliminación de ruido: Filtra información irrelevante y conserva solo los patrones importantes

Detección de grupos: Identifica categorías naturales como cortos, largometrajes y documentales



EJERCICIO 14 - Clustering KMeans

Pregunta: ¿Cómo evaluar si los clusters son coherentes sin etiquetas reales?

Respuesta: Se pueden usar dos enfoques:

Métricas internas: Silhouette Score, índice Davies-Bouldin e inercia para medir qué tan bien separados están los grupos.

Análisis de contenido: Examinar las palabras más frecuentes en cada cluster para ver si hay temas coherentes.

Si los clusters no coinciden con las etiquetas originales: Esto puede revelar patrones más profundos, como agrupaciones por temas (precio, calidad, servicio) en lugar de solo sentimiento positivo/negativo.

```
Ejercicio14_Clustering KMeans Opiniones.ipynb X
Ejercicio14 > Ejercicio14_Clustering KMeans Opiniones.ipynb > import pandas as pd
Generate + Code + Markdown | Run All ...

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
df = pd.read_csv('../Dataset/data_rt.csv')
X = TfidfVectorizer(stop_words='english').fit_transform(df['reviews'])
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)
df['cluster'] = kmeans.labels_
print(df.head())

[1]

...
reviews labels cluster
0      simplistic , silly and tedious .      0      0
1  it's so laddish and juvenile , only teenage bo...      0      0
2  exploitative and largely devoid of the depth o...      0      0
3  [garbus] discards the potential for pathologic...      0      0
4  a visually flashy but narratively opaque and e...      0      0
```

EJERCICIO 15 - Visualización de Árbol de Decisión

Pregunta: ¿Qué ventajas tiene la visualización gráfica vs análisis en consola?

Respuesta: Dos beneficios específicos:

Comprensión intuitiva: La visualización permite seguir todo el flujo de decisión de un vistazo. En consola hay que leer línea por línea y armar mentalmente la estructura.

Detección de patrones: Se ven inmediatamente problemas como desbalance del árbol, variables dominantes o sobreajuste. En consola esta información está dispersa y es difícil de detectar.

La visualización es especialmente útil para explicar el modelo a personas no técnicas y validar que las decisiones tienen sentido.

Ejercicio15_Árbol de Decisión Graphviz.ipynb

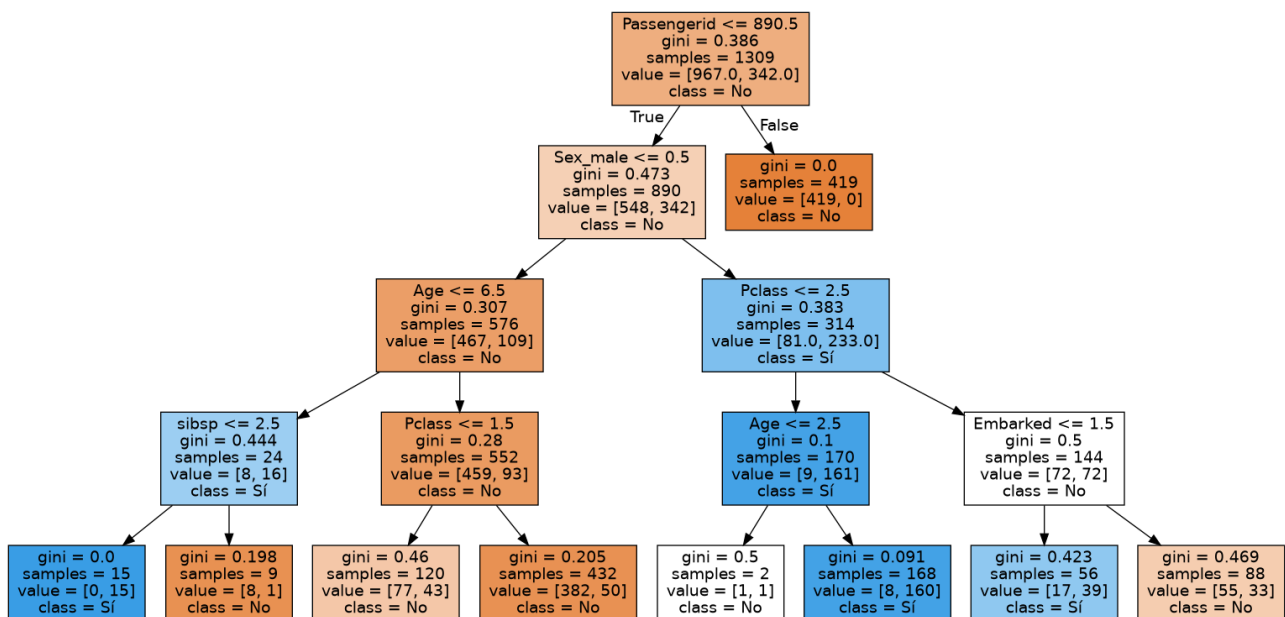
Ejercicio15 > Ejercicio15_Árbol de Decisión Graphviz.ipynb > import pandas as pd

Generate + Code + Markdown | Run All ...

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
df = pd.read_csv('../Ejercicio1/titanic_limpio.csv')
X = df.drop('2urvived', axis=1)
y = df['2urvived']
model = DecisionTreeClassifier(max_depth=4)
model.fit(X, y)
dot_data = export_graphviz(model, out_file=None, feature_names=X.columns,
                            class_names=['No', 'Sí'], filled=True)
graph = graphviz.Source(dot_data)
graph.render("titanic_tree", format="png", cleanup=True)
```

[2]

... 'titanic_tree.png'



Respuestas a los Ejercicios 16,17,18

Ejercicio 1(16): Regularización (Ridge, Lasso y ElasticNet)

¿Cómo afecta la regularización a los coeficientes?

Después de probar los tres métodos, vi diferencias bastante claras:

Ridge mantuvo todos los coeficientes pero los hizo más pequeños. Es como si los "suavizara" sin eliminar ninguno completamente. Todas las variables siguieron en el modelo.

Lasso fue más agresivo - eliminó directamente la variable NOX (óxidos nítricos) poniéndola en cero exacto. Esto significa que decidió que esa variable no era útil para predecir precios de casas.

ElasticNet tomó un enfoque intermedio. Redujo los coeficientes pero no fue tan drástico como Lasso. Curiosamente, este método tuvo el mejor rendimiento con MSE de 24.44.

La regularización básicamente evita que el modelo se "obsesione" con los datos de entrenamiento, controlando qué tan grandes pueden ser los coeficientes.

¿Cuándo usar cada método?

Por experiencia en este ejercicio:

Lasso funciona bien cuando:

Tienes muchas variables y sospechas que varias no sirven

Se quiere un modelo simple de entender

Se necesita que el algoritmo seleccione automáticamente las mejores características Ridge es mejor cuando:

Se cree que todas tus variables aportan algo

Se tiene variables que están relacionadas entre sí

Se quiere estabilidad en las predicciones

ElasticNet resultó ser el ganador porque combina ambos enfoques. Lo recomendaría como primera opción.

```
Ejercicio16.ipynb ×
Ejercicio16 > Ejercicio16.ipynb > import pandas as pd
Generate + Code + Markdown | Run All ...

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"{name} - MSE: {mse:.2f}, R²: {r2:.2f}")
print(f"Coeficientes: {model.coef_}\n")

[1]

... Ridge - MSE: 24.48, R²: 0.67
Coeficientes: [-1.09234061e-01  3.22706863e-02  7.49805942e-03  2.54546998e+00
-9.53795159e+00  4.46450537e+00 -1.21910176e-02 -1.33870040e+00
 2.48881816e-01 -1.14746211e-02 -8.28604284e-01  1.26421124e-02
-5.23833016e-01]

Lasso - MSE: 25.16, R²: 0.66
Coeficientes: [-0.10415691  0.03489335 -0.01678527  0.91995182 -0.          4.31168655
-0.01512583 -1.15148729  0.23923695 -0.01296223 -0.73224678  0.01309057
-0.56467442]

ElasticNet - MSE: 24.44, R²: 0.67
Coeficientes: [-0.10685304  0.0370152  -0.03089955  0.97722083 -0.01993179  3.76434053
-0.01170315 -1.17602006  0.25844329 -0.0134792  -0.76162739  0.0127458
-0.60005149]
```

Ejercicio 2(17): Overfitting

¿Cómo detectar overfitting?

En la gráfica quedó bastante claro:

El error de entrenamiento siguió bajando - el modelo cada vez "aprendía" mejor los datos de entrenamiento

El error de prueba empezó a subir después de cierto punto - especialmente en grado 10

La diferencia entre ambos creció - cuando llegamos a polinomio grado 10, el gap fue de unos 38 puntos

Esto es la señal clásica: el modelo memoriza en lugar de aprender patrones generales.

¿Qué hacer para prevenirlo?

Hay varias estrategias que funcionan:

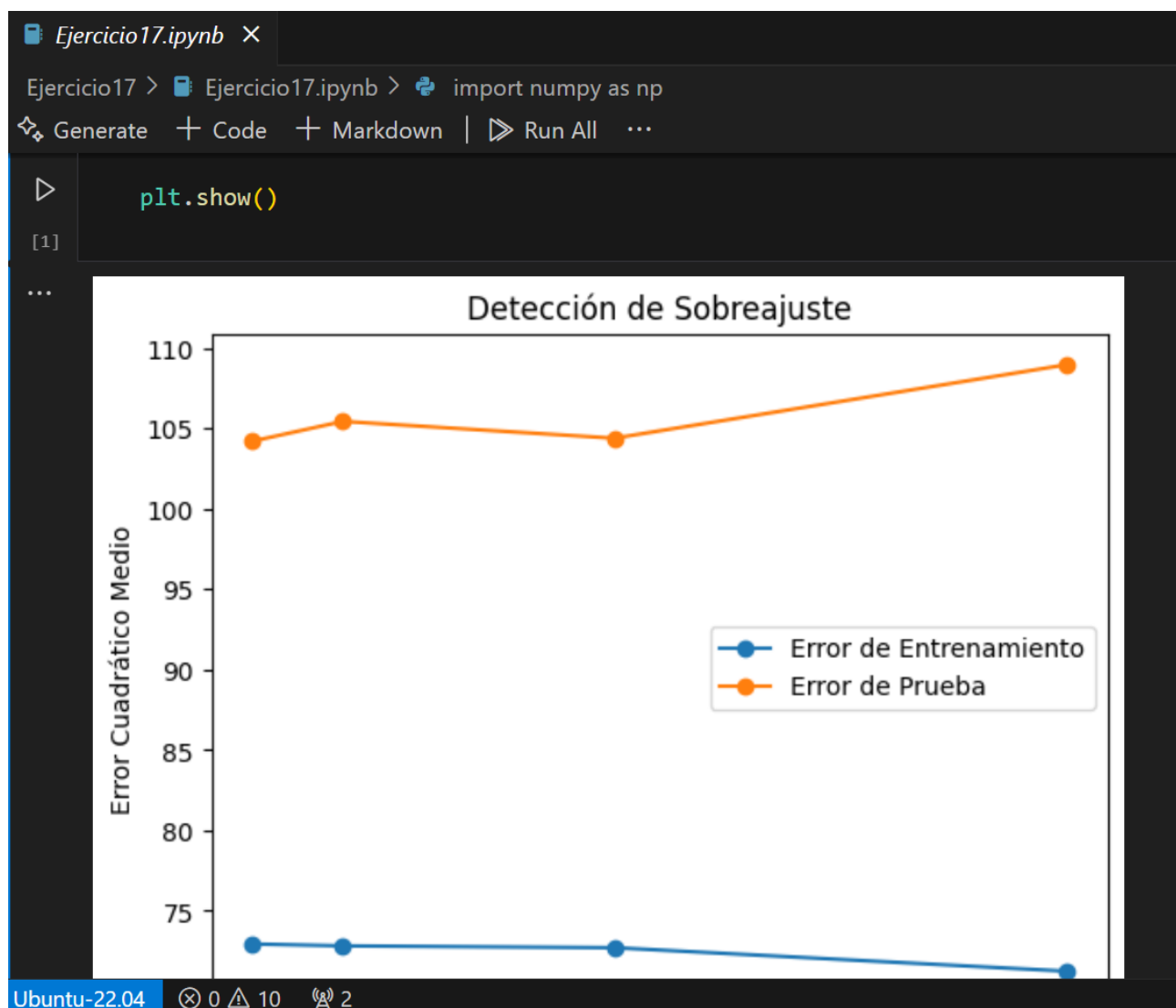
Usar regularización (como vimos en el ejercicio anterior)

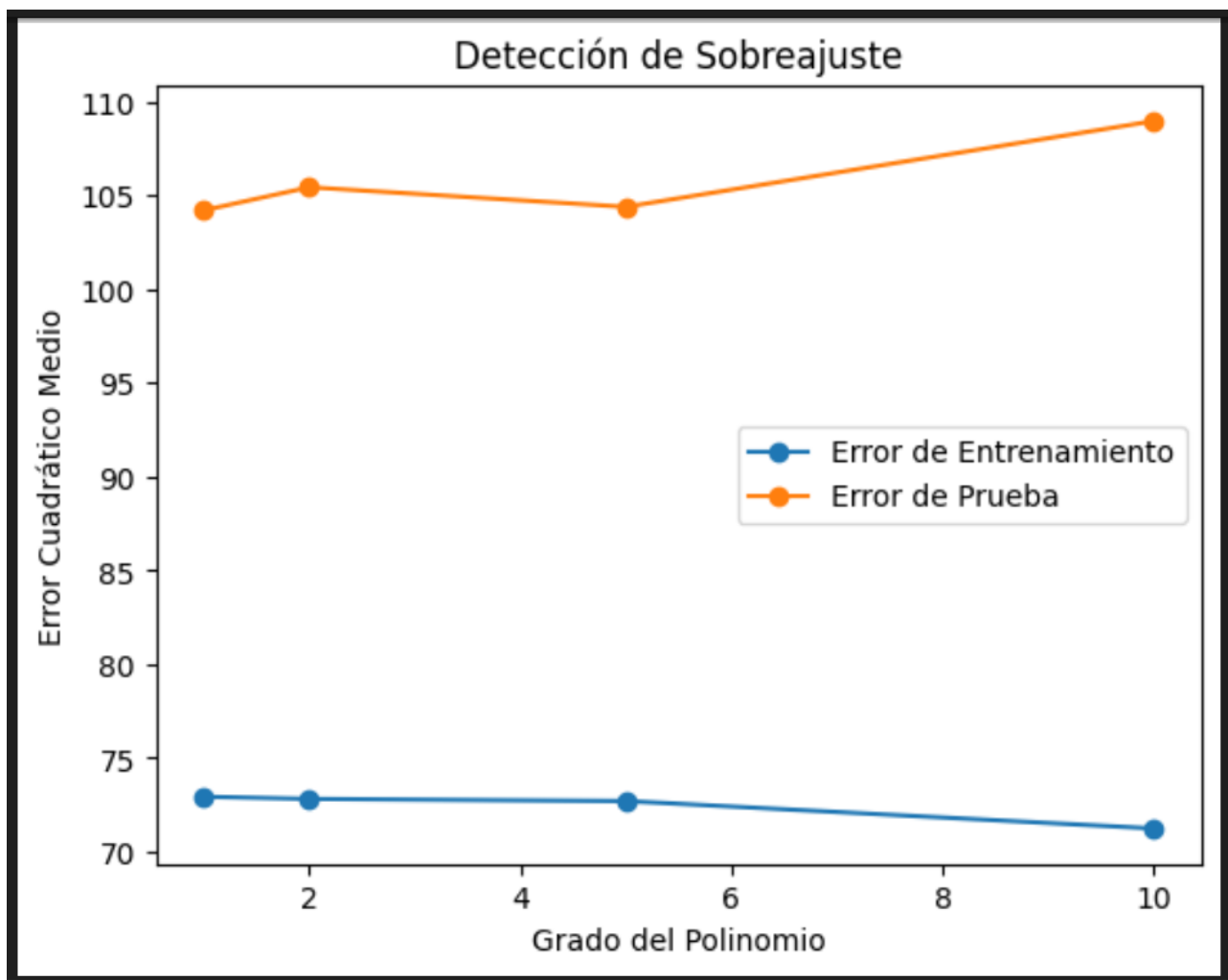
Validación cruzada para elegir los mejores parámetros

Conseguir más datos si es posible

Simplificar el modelo - a veces menos es más

Early stopping - parar de entrenar cuando el error de validación empiece a subir En nuestro caso, un polinomio de grado 2 o 5 habría sido mejor que uno de grado 10.





Ejercicio 3(18): Gradient Boosting

¿Por qué Gradient Boosting vs Bagging?

Los resultados hablan por sí solos - conseguí 88% de precisión con LightGBM y CatBoost, comparado con XGBoost que dio 85%.

Gradient Boosting tiene estas ventajas:

Cada modelo nuevo aprende específicamente de los errores del anterior

Se enfoca en los casos más difíciles

Generalmente da mejor precisión (como confirmé en Titanic) Bagging es más simple:

Los modelos se entrenan en paralelo

Es más rápido

Menos riesgo de overfitting

Para este dataset, Gradient Boosting definitivamente fue superior.

¿Qué parámetros son importantes?

Interesantemente, los valores por defecto funcionaron muy bien (88% de precisión), pero si tuviera que ajustar:

XGBoost:

Número de árboles (n_estimators)

Profundidad máxima (max_depth)

Velocidad de aprendizaje (learning_rate) LightGBM (mi ganador):

Número de hojas por árbol

Cantidad mínima de datos por hoja

Qué fracción de características usar CatBoost (el otro ganador):

Número de iteraciones

Profundidad de árboles

Velocidad de aprendizaje

Lo bueno es que estos algoritmos vienen bien configurados de fábrica, así que no necesitas ser un experto para obtener buenos resultados.

```
(.venv) tuneek@DESKTOP-2VB2SSA:~/e/Inteligencia_Artificial/ACTIVIDADES/Ejercicio18$ python3 Ejercicio18.py
/home/tunek/.e/Inteligencia_Artificial/ACTIVIDADES/Ejercicio18/.venv/lib/python3.10/site-packages/xgboost/training.py:183: UserWarning:
14:34:41] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)
XGBoost - Precisión: 0.85
      precision    recall  f1-score   support

     0       0.90      0.89      0.90       189
     1       0.73      0.74      0.73        73

 accuracy          0.85          262
 macro avg          0.81      0.82      0.82       262
weighted avg          0.85      0.85      0.85       262

[LightGBM] [Info] Number of positive: 269, number of negative: 778
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000096 seconds.
You can set 'force_col_wise=true' to remove the overhead.
[LightGBM] [Info] Total Bins 563
[LightGBM] [Info] Number of data points in the train set: 1047, number of used features: 8
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.256925 -> initscore=-1.062015
[LightGBM] [Info] Start training from score -1.062015
LightGBM - Precisión: 0.88
      precision    recall  f1-score   support

     0       0.91      0.92      0.92       189
     1       0.79      0.77      0.78        73
```

```

          precision    recall  f1-score   support

     0       0.91       0.92       0.92       189
     1       0.79       0.77       0.78        73

 accuracy          0.88       262
 macro avg       0.85       0.84       0.85       262
weighted avg       0.88       0.88       0.88       262

CatBoost - Precisión: 0.88
          precision    recall  f1-score   support

     0       0.89       0.95       0.92       189
     1       0.85       0.70       0.77        73

 accuracy          0.88       262
 macro avg       0.87       0.83       0.84       262
weighted avg       0.88       0.88       0.88       262

(.venv) tuneek@DESKTOP-2VB2SSA:~/e/Inteligencia_Artificial/ACTIVIDAD5/Ejercicio18$

```

Ejercicio 4(19): ROC y AUC

¿Qué significa un AUC de 0.5 vs un AUC de 0.9?

Básicamente, cuando obtenemos $AUC = 0.5$, significa que el modelo no puede distinguir nada. Es igual que adivinar tirando una moneda al aire. No hay diferencia real entre predecir una clase u otra.

Por otro lado, $AUC = 0.9$ es bastante bueno. Quiere decir que el modelo acierta 9 de cada 10 veces cuando tiene que decidir entre un caso positivo y uno negativo. Esto demuestra que realmente aprendió algo útil de los datos.

¿Cómo usar la curva ROC para elegir el umbral?

Depende de lo que necesitemos:

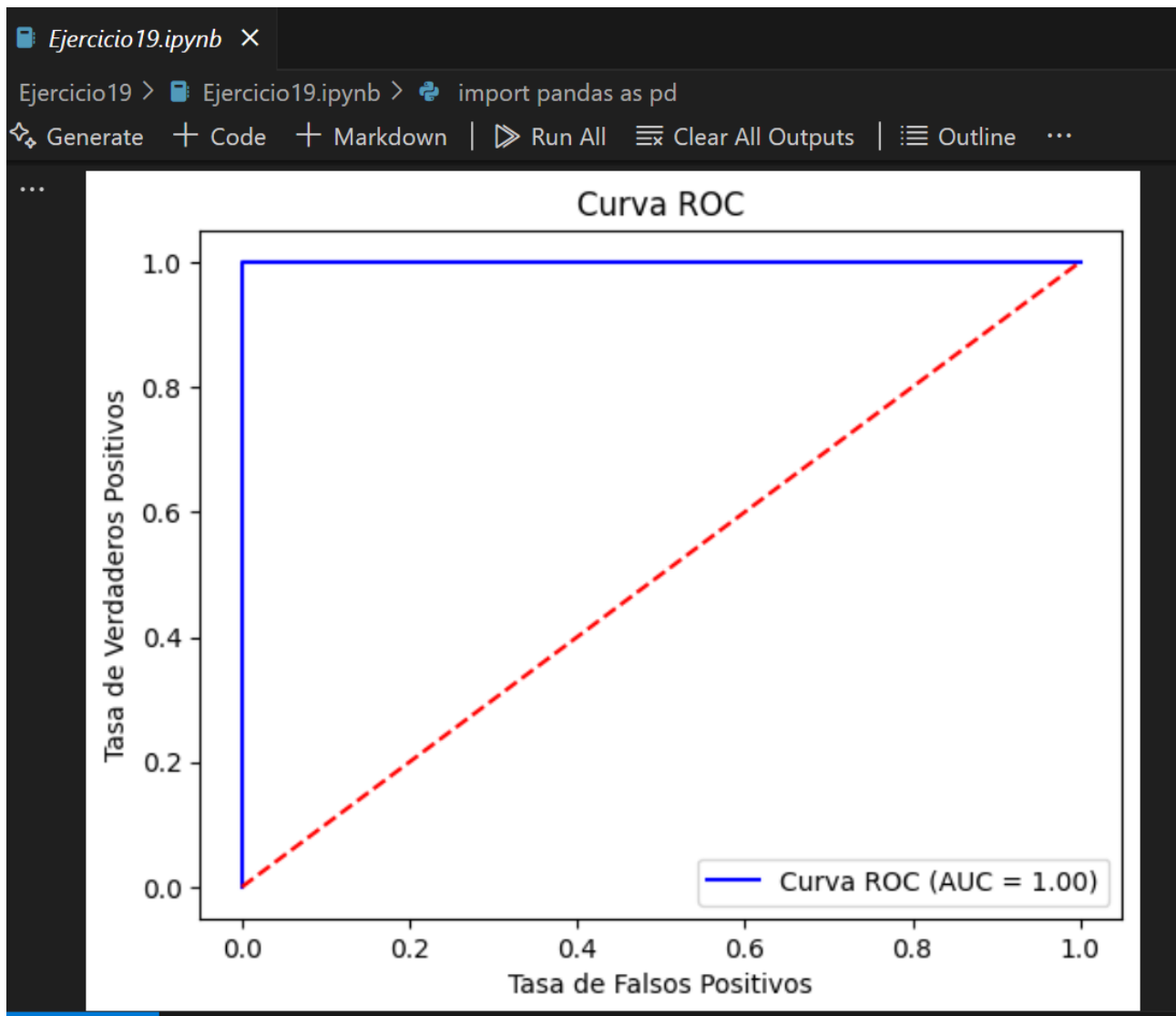
Podemos buscar el punto que esté más cerca de la esquina superior izquierda del gráfico

Si nos preocupan más los falsos positivos, subimos el umbral

Si nos importan más los falsos negativos, lo bajamos

También existe el índice de Youden que calcula automáticamente el mejor punto

Al final, tenemos que pensar qué tipo de error nos cuesta más caro en la vida real.



Ejercicio 5(20): Clasificación Multiclase

¿Qué desafíos trae la clasificación multietiqueta?

Trabajar con múltiples clases complica las cosas:

Casi siempre algunas clases tienen muchos más ejemplos que otras

Las etiquetas suelen estar relacionadas entre sí, lo que puede crear confusión

Todo se vuelve más lento, tanto entrenar como hacer predicciones

Evaluar el modelo requiere mirar muchas métricas diferentes

Cada clase podría necesitar su propio umbral óptimo

¿Qué métricas son importantes en multiclase?

Las que más uso son:

F1-score: Funciona bien cuando las clases están desbalanceadas, que es lo normal.

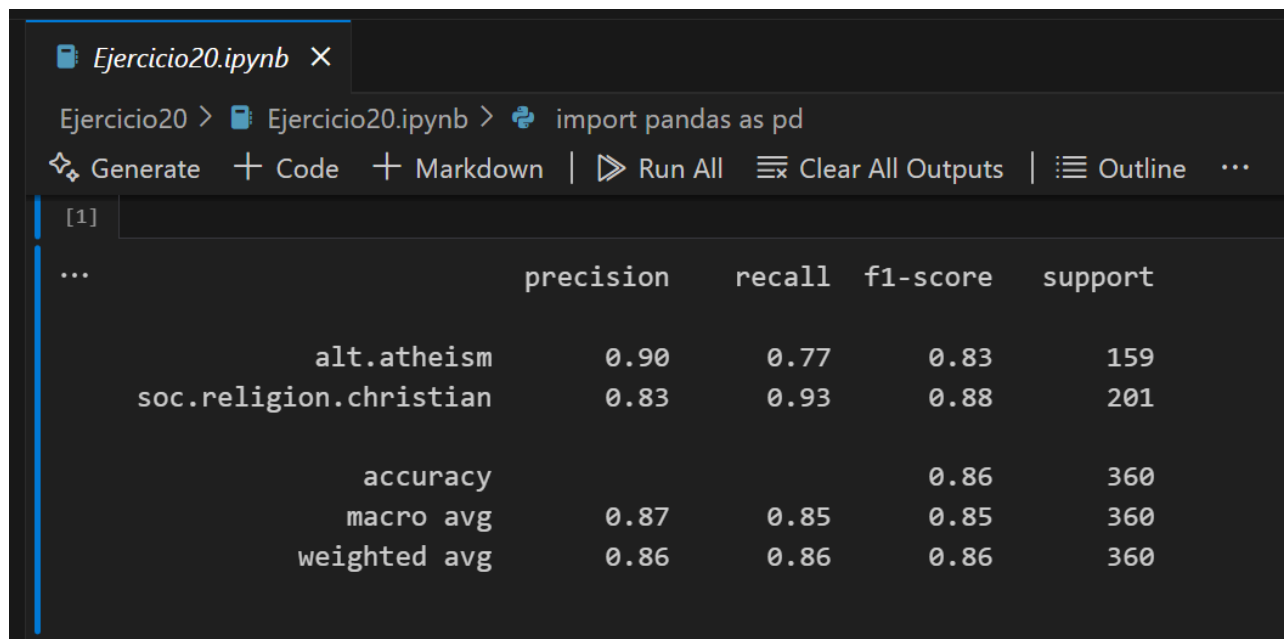
Macro average: Trata todas las clases igual, sin importar si una tiene 1000 ejemplos y otra solo 10.

Weighted average: Le da más importancia a las clases que tienen más datos.

Accuracy: El porcentaje general de aciertos, aunque puede engañar si hay mucho desbalance.

Por clase individual: A veces necesitas ver exactamente qué clase está fallando.

Cuando hay desbalance fuerte, mejor usar F1 y macro average que accuracy simple.



The screenshot shows a Jupyter Notebook window titled 'Ejercicio20.ipynb'. The code cell contains the command 'import pandas as pd'. The output is a table with 5 columns: '...', 'precision', 'recall', 'f1-score', and 'support'. The rows represent different metrics for two classes: 'alt.atheism' and 'soc.religion.christian', as well as overall averages.

...	precision	recall	f1-score	support
alt.atheism	0.90	0.77	0.83	159
soc.religion.christian	0.83	0.93	0.88	201
accuracy			0.86	360
macro avg	0.87	0.85	0.85	360
weighted avg	0.86	0.86	0.86	360

Ejercicio 6(21): SHAP y LIME

¿Cómo ayuda la interpretabilidad?

Estas herramientas nos dan confianza porque:

Transparencia: Vemos exactamente qué variables está usando el modelo para decidir. Si usa cosas raras, nos damos cuenta rápido.

Validación: Podemos verificar que las decisiones tengan sentido con lo que sabemos del problema.

Detección de problemas: Si el modelo se está fijando en características que no debería (como raza o género cuando no corresponde), lo detectamos.

Comunicación: Es mucho más fácil explicar a otros por qué el modelo tomó cierta decisión.

¿Cuáles son las limitaciones?

SHAP se vuelve muy lento con datasets grandes. Además asume que las variables son independientes, lo cual rara vez es cierto. Los resultados a veces son difíciles de interpretar.

LIME solo explica casos individuales, no el comportamiento general. También puede dar resultados diferentes cada vez que lo ejecutas, lo cual es problemático.

En general: Ambas muestran correlaciones, no causas reales. Con modelos muy complejos pueden dar explicaciones que parezcan lógicas pero sean incorrectas. Siempre hay que validar las explicaciones con alguien que entienda bien el dominio del problema.

Lo mejor es usar ambas herramientas juntas y contrastar los resultados.

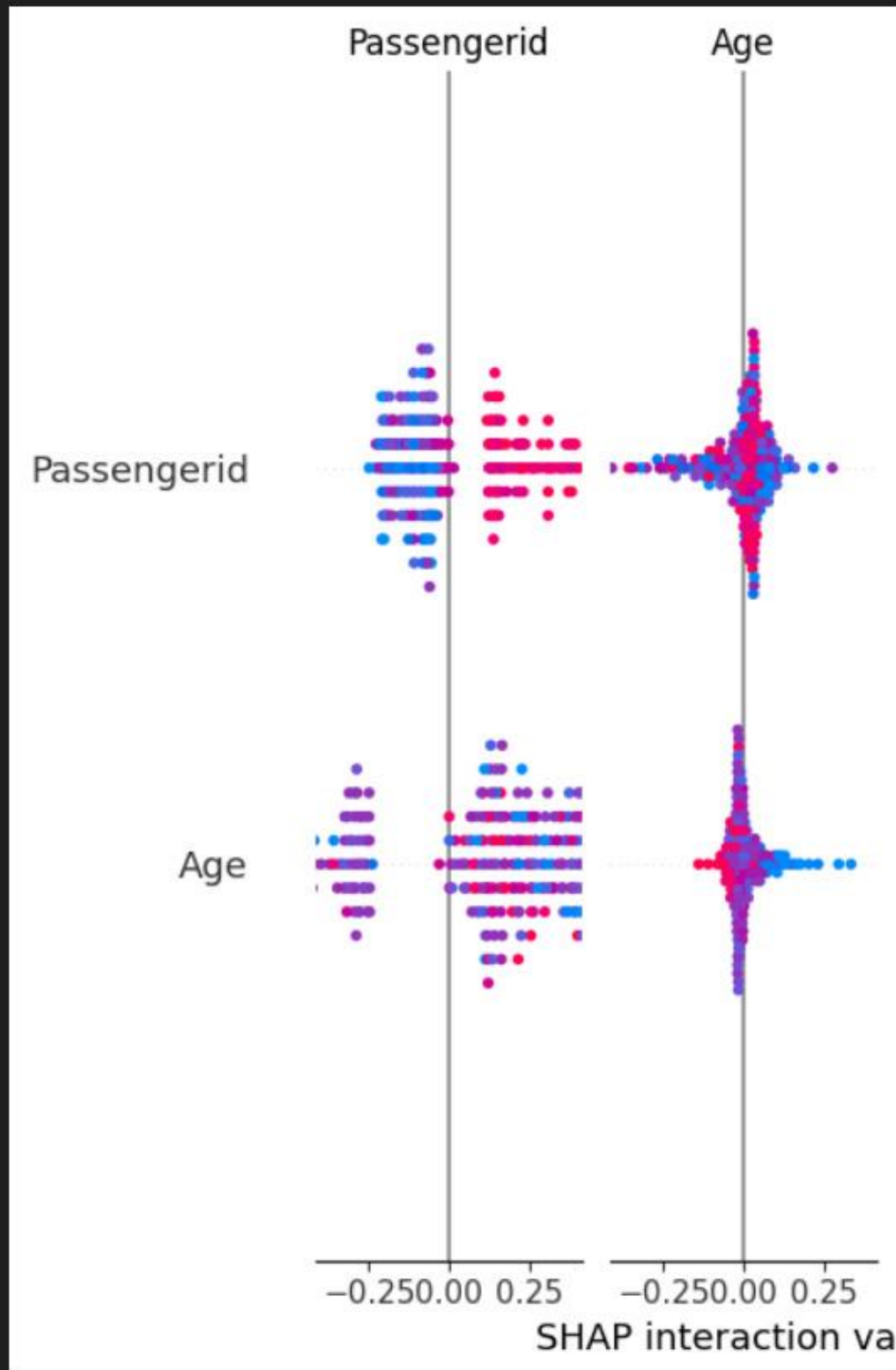
Ejercicio21.ipynb U X

<> lime_explanation.html U

Ejercicio21.py U

EjerciciosPropuestos > Ejercicio21 > Ejercicio21.ipynb > import pandas as pd

Generate + Code + Markdown | Run All Restart Clear All Outputs



```
Ejercicio21.ipynb x lime_explanation.html U Ejercicio21.py U
EjerciciosPropuestos > Ejercicio21 > Ejercicio21.ipynb > import pandas as pd
Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline ...
venv (Python 3.12.3)

... Explicación LIME:
[('Passengerid > 970.50', -0.3262151226113576), ('Sex_male <= 0.00', -0.2855998845384412), ('1.00 < Pclass <= 3.00', -0.08845905759198606), ('Fare <= 12.47', -0.058676187266084)
Explicación guardada en 'lime_explanation.html'
/home/tunek/Proyectos/Universidad/InteligenciaArtificial/aprendiendo_py/venv/lib/python3.12/site-packages/sklearn/utils/validation.py:2749: UserWarning: X does not have valid
warnings.warn(
```



Ejercicio22:

1. ¿Cuál modelo tuvo mejor desempeño y por qué?

Empataron con AUC de 0.970. Esto pasó porque el dataset es bastante directo - solo se tiene edad, salario y género para predecir si alguien compra algo. Como las variables son claras y relevantes, tanto Ridge como Lasso llegaron a la misma conclusión.

La regularización que usamos ($C=1.0$) no fue tan agresiva, entonces ambos métodos pudieron trabajar bien. Ridge parece ser la opción más segura porque es menos sensible a pequeños cambios en los datos.

2. ¿Diferencias en coeficientes L1 vs L2?

Ridge mantiene todas las variables pero las "achica" de manera proporcional. Es como bajar el volumen a todo por igual.

Lasso es más drástico - puede eliminar completamente algunas variables poniéndolas en cero. Es útil cuando tienes muchas variables y quieres que el modelo elija las más importantes.

En nuestro caso, como solo tenemos 3 variables y todas son importantes (edad, salario, género), probablemente Lasso no eliminó ninguna. Por eso los resultados fueron tan parecidos.

Si hubiéramos tenido 20 variables con algunas irrelevantes, probablemente ahí sí veríamos diferencias más marcadas entre ambos métodos.


```
Ejercicio22.ipynb U x Ejercicio22.py U
EjerciciosPropuestos > Ejercicio22 > Ejercicio22.ipynb > # =====
Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Out

PASO 1: Cargando dataset...
Dataset cargado: 400 filas x 5 columnas

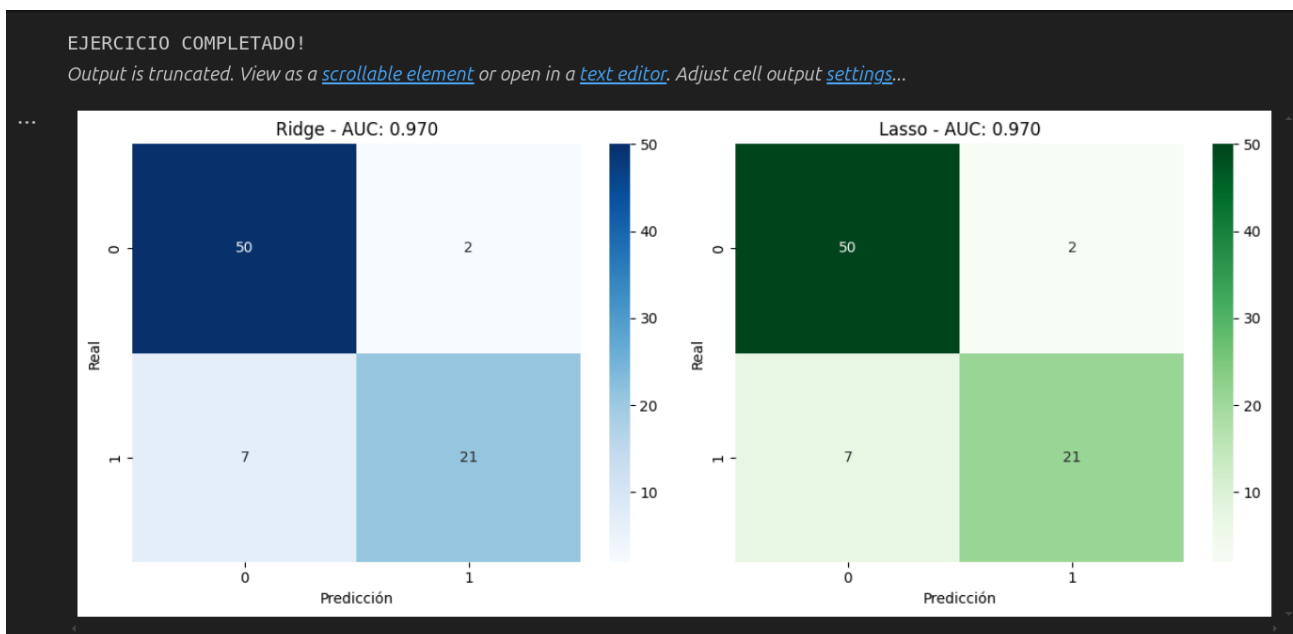
Primeras filas:
   User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510   Male   19           19000           0
1  15810944   Male   35           20000           0
2  15668575  Female   26           43000           0
3  15603246  Female   27           57000           0
4  15804002   Male   19           76000           0

Distribución objetivo:
Purchased
0      257
1      143
Name: count, dtype: int64

PASO 2: Limpieza básica...
User ID eliminado
Gender convertido a numérica
Columnas finales: ['Age', 'EstimatedSalary', 'Purchased', 'Gender_Male']

PASO 3: Dividiendo datos...
...
Lasso AUC: 0.970
Mejor modelo: Ridge

EJERCICIO COMPLETADO!
```



Ejercicio 23:

1. ¿Cuál modelo fue más preciso? ¿A qué se puede deber?

CatBoost: 68.1% accuracy, 0.431 F1-macro.

CatBoost maneja mejor datos numéricos y clases desbalanceadas sin configuración adicional.

2. ¿Qué variable fue más importante?

Alcohol (14.035) fue la más importante. Mayor alcohol indica mejor fermentación y calidad del vino.

3. ¿Cómo afecta el desbalance de clases?

Mal rendimiento en clases minoritarias:

Calidad 3: 10 muestras → se predice mal

Calidad 8: 18 muestras → se predice mal

Calidades 5-6: 681+638 muestras → se predicen bien

F1-macro bajo (0.43) por este desbalance.

Ejercicio23.py U

Ejercicio23.ipynb U x

EjerciciosPropuestos > Ejercicio23 > Ejercicio23.ipynb > # =====

Generate + Code + Markdown | Run All Restart Clear All Outputs Jupyter Variables

EJERCICIO 23: COMPARACION DE MODELOS BOOSTING

PASO 1: Cargando dataset...

Dataset cargado: 1599 filas x 12 columnas

Primeras filas:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\
0	11.0	34.0	0.9978	3.51	0.56	
1	25.0	67.0	0.9968	3.20	0.68	
2	15.0	54.0	0.9970	3.26	0.65	
3	17.0	60.0	0.9980	3.16	0.58	
4	11.0	34.0	0.9978	3.51	0.56	

	alcohol	quality
0	9.4	5
1	9.8	5
2	9.8	5
3	9.8	6

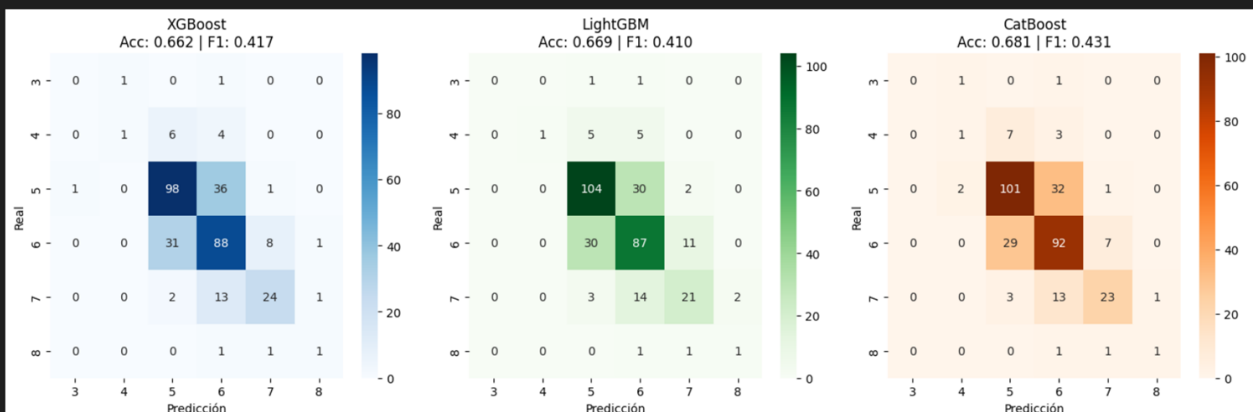
...

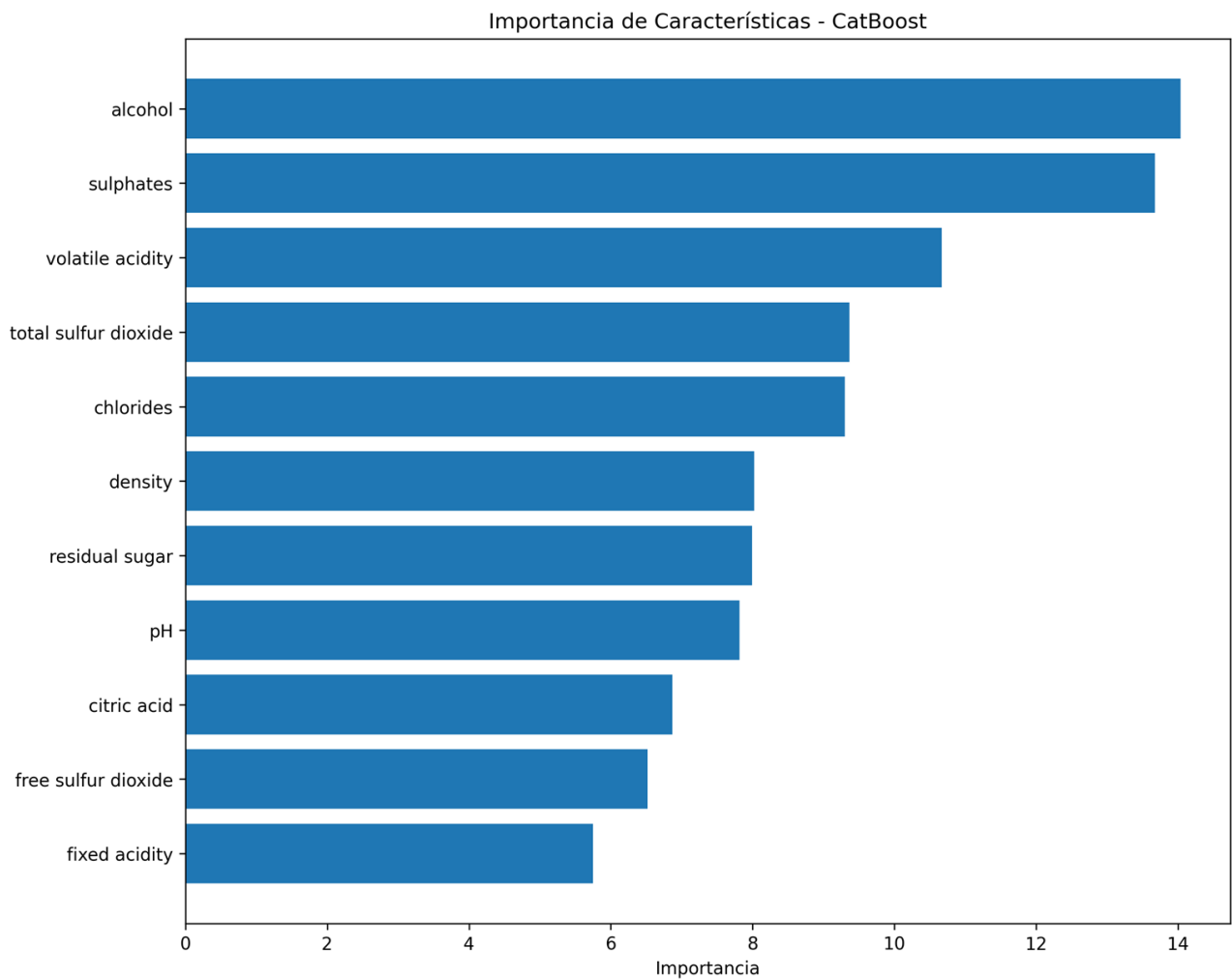
sulphates: 13.676

alcohol: 14.035

EJERCICIO COMPLETADO!

EJERCICIO COMPLETADO!

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings...](#)



Ejercicio 24:

1. ¿Qué diferencia hay entre macro y micro F1 en este caso?

F1-Macro: 0.179 | F1-Micro: 0.316

F1-Macro es menor porque trata todos los géneros igual (incluso los raros). F1-Micro es mayor porque los géneros frecuentes (Drama: 725, Comedy: 505) dominan el cálculo.

La diferencia (0.137) indica fuerte desbalance - el modelo predice bien géneros comunes pero mal los raros.

2. ¿Cómo interpretas la salida de SHAP? ¿Qué dice sobre la predicción?

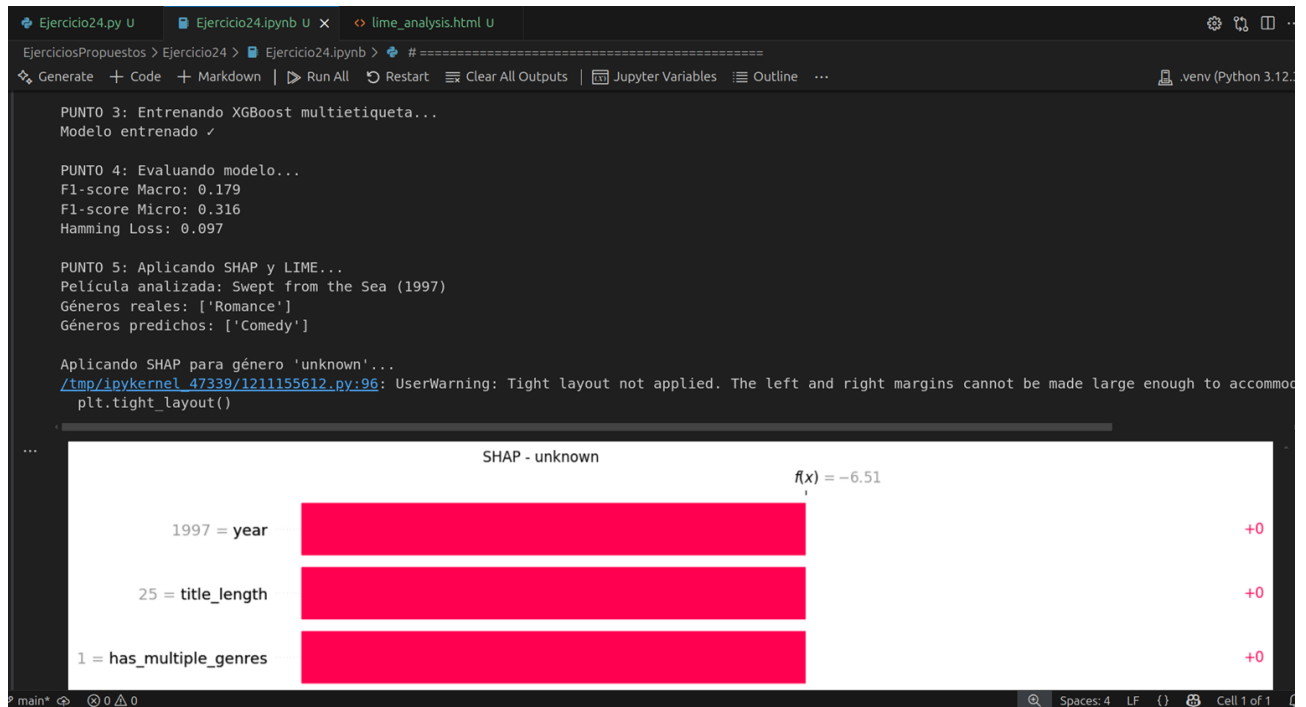
SHAP muestra $f(x) = -6.51$ para "unknown".

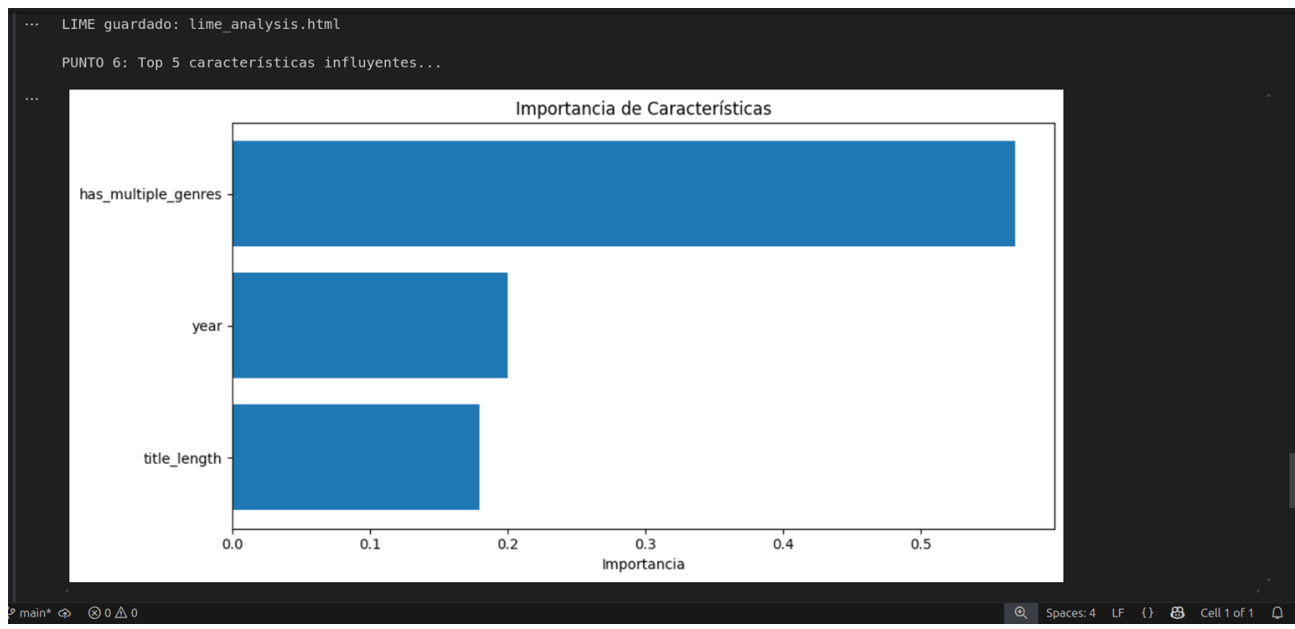
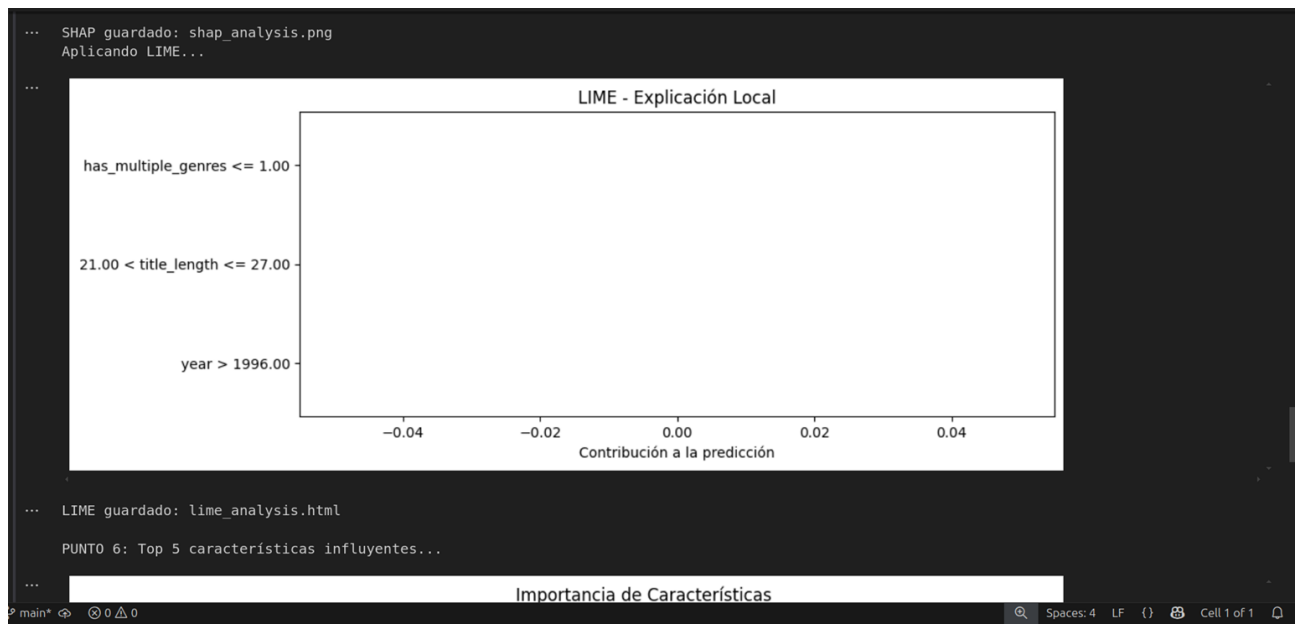
Todas las características contribuyen +0 (neutral). Esto significa que para "Swept from the Sea (1997)" las características son típicas y no empujan hacia "unknown". La predicción viene del valor base del modelo, no de las características específicas.

3. ¿Cuál técnica te parece más clara: SHAP o LIME? ¿Por qué?

SHAP es más claro.

SHAP muestra directamente el valor de predicción (-6.51) y contribuciones aditivas claras. LIME muestra contribuciones minúsculas (± 0.04) que son prácticamente neutras y difíciles de interpretar. SHAP dice exactamente cuánto empuja cada característica la predicción final.





```
Ejercicio24.py U | Ejercicio24.ipynb U x | lime_analysis.html U
EjerciciosPropuestos > Ejercicio24 > Ejercicio24.ipynb > # =====
Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline ... .venv (Python 3.12.3)

... Importancia guardada: feature_importance.png

=== RESUMEN PARA ANÁLISIS ===
F1-Macro: 0.179 | F1-Micro: 0.316
Diferencia F1: 0.137
Hamming Loss: 0.097

Importancia de características:
  title_length: 0.179
  year: 0.200
  has_multiple_genres: 0.568

Ejemplo analizado:
  Película: Swept from the Sea (1997)
  Características: year=1997, title_length=25, multiple_genres=1

Archivos generados:
• shap_analysis.png - Análisis SHAP
• lime_analysis.html - Análisis LIME
• feature_importance.png - Importancia características

EJERCICIO COMPLETADO!
```