UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES
DEPARTAMENTO DE COMPUTACIÓN Y SIMULACIÓN DE SISTEMAS

TÓPICOS ESPECIALES I

TALLER 4

Facilitador: Tomás J. Concepción Miranda

Integrantes:

Gustavo Colucci 8-951-2191
Luis Mejia 8-949-350
Greg Torres 8-956-675

Objetivo

Aprender las funciones y métodos fundamentales de la librería scikit-learn.

Indicaciones

Se debe realizar un informe en el que se detalle, para cada problema, el desarrollo de la solución. Este trabajo se puede realizar en grupos de 2 o 3 personas. Envíe su informe en Moodle en formato PDF, así como el código fuente, en el bloque correspondiente antes de las 11:55 p.m. del día siguiente del taller

Rúbrica

Este laboratorio tendrá una puntuación total de 100, donde la evaluación se basada en los aspectos de excelente, bueno, regular, deficiente. Los puntos que se evaluará en la rúbrica se muestran en la tabla:

N.	Aspectos a evaluar				
1	**CONTENIDO DE ACUERDO A LO SOLICITADO EN EL ENUNCIADO**	Excelente (80)	Bueno (70)	Regular (50)	Deficiente (0)
	Resultados debidamente presentados (capturas de pantalla, explicación) - (80 puntos)				
2	**PANTALLA DE PRESENTACIÓN -** (10 puntos)	Excelente (10)	Bueno (8)	Regular (5)	Deficiente (0)
	Cumple con todos los parámetros dados en clase				
3	**ENTREGA DE TRABAJO EN LA PLATAFORMA MOODLE** - (10 puntos)	Excelente (10)	Bueno (8)	Regular (5)	Deficiente (0)
	Entrega a tiempo en la plataforma				

Enunciados

Información General

scikit-learn es una librería que provee “herramientas simples y eficientes para el analysis predictivo de datos”. Entre las diferentes herramientas que tiene, se puede mencionar métodos de clasificación como máquinas de vector de soporte (SVM), kvecions más cercanos (kNN), arbol de decisión; métodos de regresión como mínimos cuadrados ordinarios, Lasso; y métodos de agrupamiento como k-medios (k−means), agrupamiento herárquico. También cuenta técnicas de preprocesmiento y transformación de juegos de datos, así como formas de visualizar las fronteras de decisión de los modelos creados.

La documentación oficial se encuentra disponible en scikit-learn.org/stable/modules/classes.html

Existen ejemplos disponibles por los desarrolladores de la librería

Problema 1:

Detalle del problema:

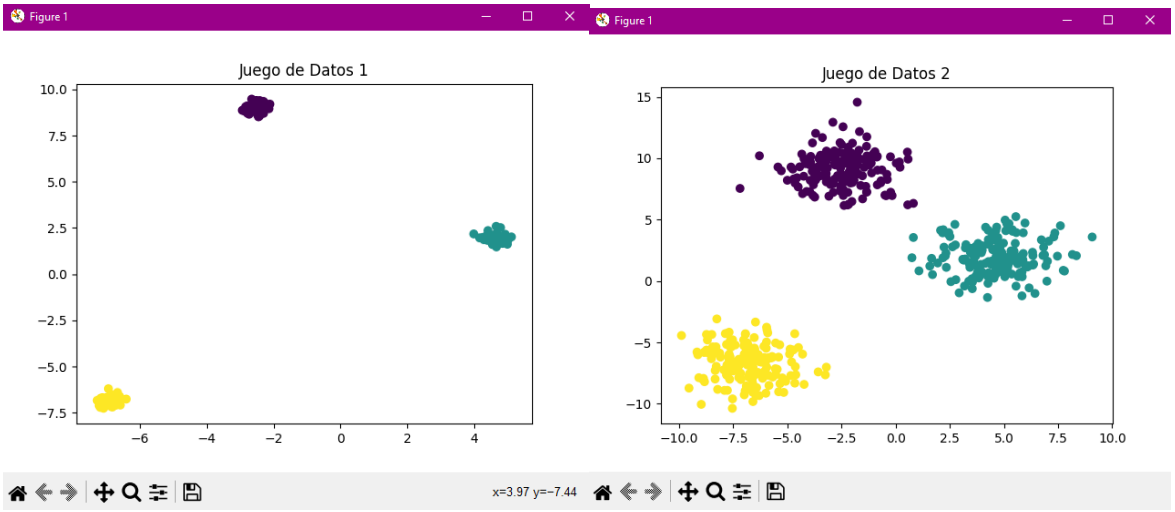
Crear dos juegos de datos usando `make_blobs` con los siguientes parámetros y visualizarlos:

1.Tamaño de muestra: 100, varianza de 0.25

2. Tamaño de muestra: 500, varianza de 1.44

```
#Problema 1
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Crear el primer juego de datos
X1, y1 = make_blobs(n_samples=100, cluster_std=0.25, random_state=42)
plt.scatter(X1[:, 0], X1[:, 1], c=y1)
plt.title("Juego de Datos 1")
plt.show()

# Crear el segundo juego de datos
X2, y2 = make_blobs(n_samples=500, cluster_std=1.44, random_state=42)
plt.scatter(X2[:, 0], X2[:, 1], c=y2)
plt.title("Juego de Datos 2")
plt.show()
```

Imagen:



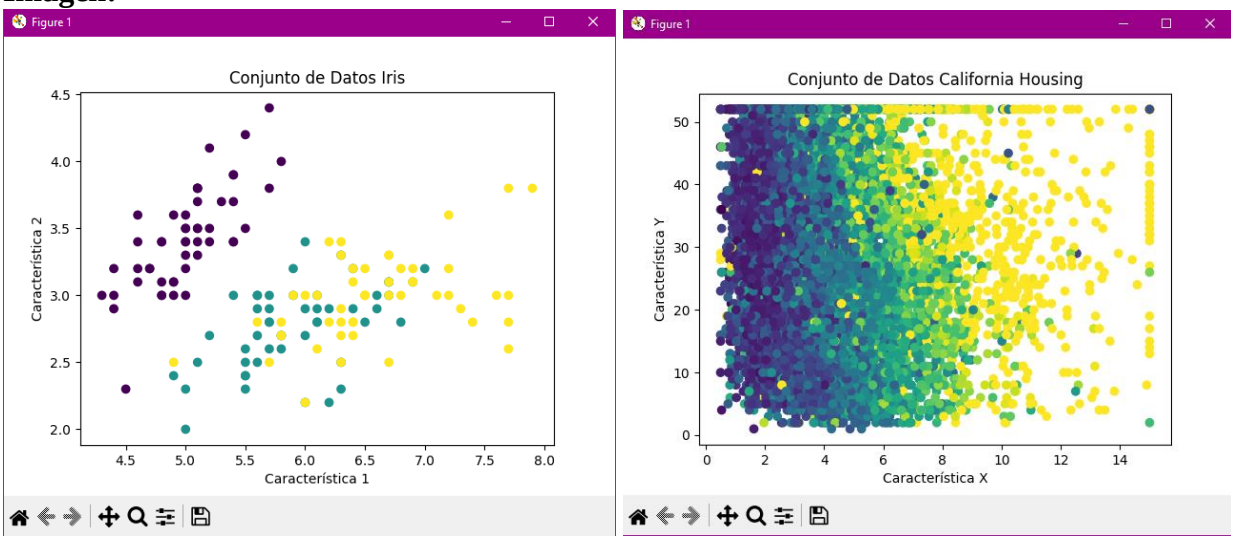
Explicación: hemos utilizado la función make_blobs de scikit-learn para crear dos juegos de datos sintéticos con diferentes tamaños de muestra y varianzas. Luego, hemos visualizado estos juegos de datos utilizando gráficos de dispersión. El primer juego de datos (Juego de Datos 1) tiene una varianza baja, lo que significa que los puntos están más agrupados, mientras que el segundo juego de datos (Juego de Datos 2) tiene una varianza más alta, lo que significa que los puntos están más dispersos.

Problema 2:

Detalle del problema: Cargar el dataset iris en una variable iris y el dataset fetch_california_housing en una variable california.

```
from sklearn.datasets import load_iris, fetch_california_housing

import matplotlib.pyplot as plt

# Cargar el dataset iris
iris = load_iris()

# Cargar el dataset California housing
california = fetch_california_housing()
```

Imagen:



Explicación:
Hemos cargado dos conjuntos de datos diferentes utilizando las funciones load_iris y fetch_california_housing de scikit-learn. El conjunto de datos "iris" se utiliza comúnmente para problemas de clasificación, mientras que el conjunto de datos "California housing" se utiliza para problemas de regresión.

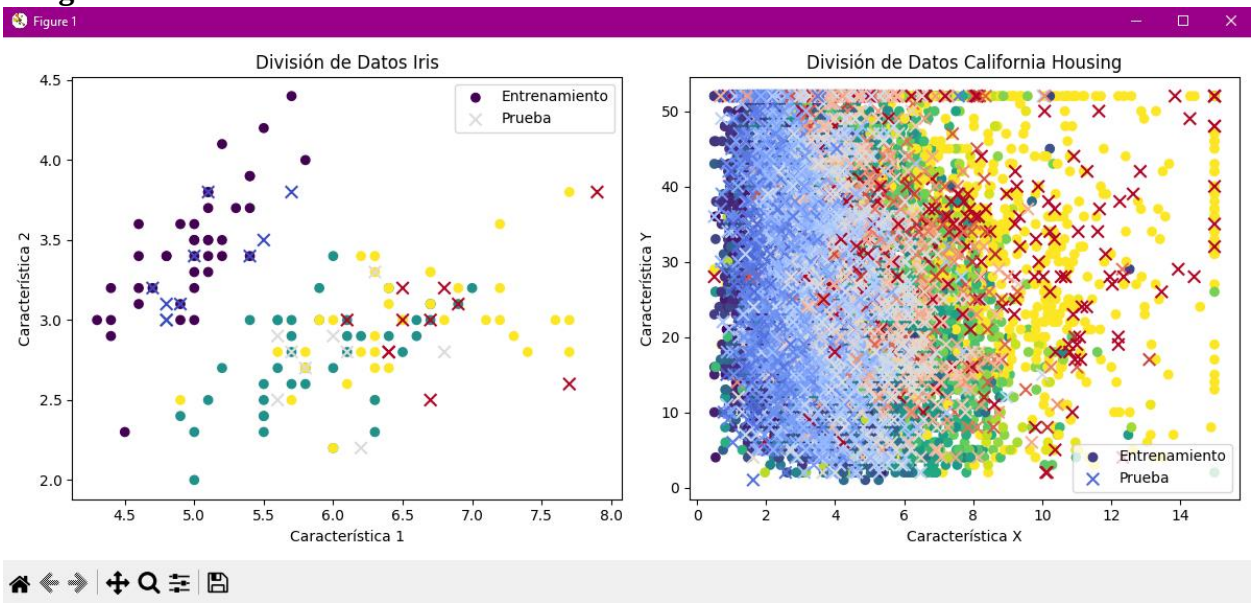
Problema 3:

Detalle del problema: Divida los juegos de datos iris y california en 80 % entrenamiento y 20 % prueba.

```
from sklearn.model_selection import train_test_split

# Dividir el dataset iris en entrenamiento y prueba
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(
    iris.data, iris.target, test_size=0.2, random_state=42)

# Dividir el dataset California en entrenamiento y prueba
X_train_california, X_test_california, y_train_california, y_test_california =
train_test_split(
    california.data, california.target, test_size=0.2, random_state=42)

# Visualizar la división de datos de iris en conjuntos de entrenamiento y prueba
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_train_iris[:, 0], X_train_iris[:, 1], c=y_train_iris,
cmap='viridis', label='Entrenamiento')
plt.scatter(X_test_iris[:, 0], X_test_iris[:, 1], c=y_test_iris, cmap='coolwarm',
marker='x', s=70, label='Prueba')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.title('División de Datos Iris')
plt.legend()

# Visualizar la división de datos de California en conjuntos de entrenamiento y
prueba
plt.subplot(1, 2, 2)
plt.scatter(X_train_california[:, 0], X_train_california[:, 1],
c=y_train_california, cmap='viridis', label='Entrenamiento')
plt.scatter(X_test_california[:, 0], X_test_california[:, 1],
c=y_test_california, cmap='coolwarm', marker='x', s=70, label='Prueba')
plt.xlabel('Característica X')
plt.ylabel('Característica Y')
plt.title('División de Datos California Housing')
plt.legend()

plt.tight_layout()
```

```
plt.show()
```

Imagen:



Explicación: En este problema, hemos utilizado la función train_test_split de scikit-learn para dividir los conjuntos de datos "iris" y "California housing" en conjuntos de entrenamiento y prueba. Hemos especificado que el 80% de los datos se utilizarán para entrenamiento y el 20% se utilizará para pruebas. Esto es esencial para evaluar el rendimiento de los modelos

Problema 4:

Detalle del problema: Cree un estimador SVC en la variable clf_svc

```
from sklearn.svm import SVC

# Crear un estimador SVC
clf_svc = SVC()
```

Explicación: En este problema, estamos creando un estimador de clasificación de Máquinas de Soporte Vectorial (SVC, por sus siglas en inglés) y lo estamos almacenando en una variable llamada clf_svc.

SVC es una clase de scikit-learn que representa un clasificador de Máquinas de Soporte Vectorial. Al crear una instancia de SVC() estamos inicializando un modelo SVC con los hiperparámetros predeterminados.
El modelo SVC se utiliza comúnmente para problemas de clasificación y es capaz de encontrar fronteras de decisión no lineales en los datos. Una vez que hemos creado el estimador clf_svc, podemos utilizarlo para entrenar el modelo con datos de entrenamiento y realizar predicciones en datos de prueba.

Problema 5:

Detalle del problema:

```
#problema 5
# Entrenar el estimador SVC con los datos de entrenamiento de iris
clf_svc.fit(X_train_iris, y_train_iris)
```

Explicación: X_train_iris: Representa las características de entrenamiento del conjunto de datos de iris. Estas características son los atributos que se utilizan para aprender a clasificar las muestras.

y_train_iris: Representa las etiquetas de entrenamiento correspondientes al conjunto de datos de iris. Estas etiquetas son las categorías correctas a las que pertenecen las muestras de entrenamiento.

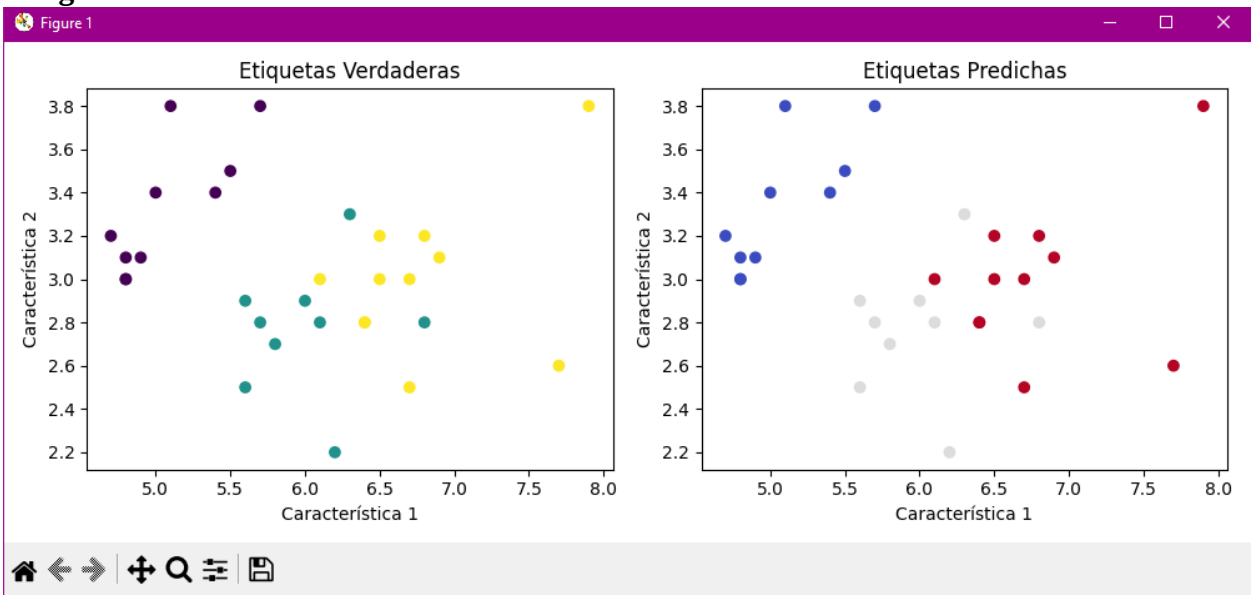
Problema 6:

Detalle del problema: Hacer la predicción de los datos de prueba usando el estimador clf_svc con el método predict. Comparar la predicción con los valores verdaderos y mostrarlo gráficamente.

```
#problema 6
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Hacer predicciones en el conjunto de prueba de iris
y_pred_iris = clf_svc.predict(X_test_iris)

# Calcular la exactitud (accuracy) de las predicciones
accuracy = accuracy_score(y_test_iris, y_pred_iris)

# Crear una figura Matplotlib para la comparación
plt.figure(figsize=(10, 4))

# Gráfico 1: Mostrar las etiquetas verdaderas
plt.subplot(1, 2, 1)
plt.scatter(X_test_iris[:, 0], X_test_iris[:, 1], c=y_test_iris, cmap='viridis')
plt.title('Etiquetas Verdaderas')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
# Gráfico 2: Mostrar las etiquetas predichas por el modelo
plt.subplot(1, 2, 2)
plt.scatter(X_test_iris[:, 0], X_test_iris[:, 1], c=y_pred_iris, cmap='coolwarm')
plt.title('Etiquetas Predichas')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')

plt.tight_layout()
plt.show()
```

```
# Calcular la exactitud y mostrarla
print(f'Exactitud del modelo SVC en el conjunto de prueba: {accuracy:.2f}')
```

Imagen:



Explicación:

En este problema, primero utilizamos el modelo SVC (clf_svc) que entrenamos previamente para hacer predicciones en el conjunto de datos de prueba de iris (X_test_iris). Las etiquetas predichas se almacenan en la variable y_pred_iris.

Luego, hemos calculado la exactitud (accuracy) de las predicciones utilizando la función accuracy_score de scikit-learn. La exactitud es una métrica que mide la proporción de predicciones correctas en comparación con las etiquetas verdaderas.

Para visualizar la comparación entre las etiquetas verdaderas y las etiquetas predichas, hemos creado una figura Matplotlib con dos gráficos. El primer gráfico muestra las etiquetas verdaderas como colores en función de las características, mientras que el segundo gráfico muestra las etiquetas predichas por el modelo. Esto permite una comparación visual de cómo se ajustan las predicciones a los datos reales.

Finalmente, mostramos la exactitud del modelo en el conjunto de prueba. Esto proporciona una medida cuantitativa de qué tan bien el modelo SVC está haciendo sus predicciones en comparación con las etiquetas verdaderas.

Problema 7:

Detalle del problema: Calcular la exactitud, precisión, recuperación y valor-F1 del estimador usando las etiquetas verdaderas y las etiquetas estimadas.

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

# Calcular la exactitud
accuracy = accuracy_score(y_test_iris, y_pred_iris)

# Calcular la precisión
precision = precision_score(y_test_iris, y_pred_iris, average='weighted')

# Calcular la recuperación
recall = recall_score(y_test_iris, y_pred_iris, average='weighted')

# Calcular el valor-F1
f1 = f1_score(y_test_iris, y_pred_iris, average='weighted')

# Mostrar las métricas calculadas
print(f'Exactitud: {accuracy:.2f}')
print(f'Precisión: {precision:.2f}')
print(f'Recuperación: {recall:.2f}')
print(f'Valor-F1: {f1:.2f}')
```

Explicación:

En este problema, calculamos cuatro métricas de evaluación diferentes para evaluar el rendimiento del modelo SVC en el conjunto de prueba de iris. Estas métricas son:

Exactitud (Accuracy): Mide la proporción de predicciones correctas en relación con el total de predicciones realizadas.

Precisión (Precision): Mide la proporción de verdaderos positivos (muestras correctamente clasificadas como positivas) en relación con el total de muestras clasificadas como positivas por el modelo.

Recuperación (Recall): Mide la proporción de verdaderos positivos en relación con el total de muestras reales que son positivas.

Valor-F1 (F1-Score): Es una métrica que combina precisión y recuperación en una sola puntuación. Es especialmente útil cuando hay un desequilibrio en las clases.

Utilizamos las funciones accuracy_score, precision_score, recall_score y f1_score de scikit-learn para calcular estas métricas. El argumento average='weighted' se utiliza para calcular las métricas teniendo en cuenta la ponderación de las clases, lo que es útil cuando hay desequilibrio de clases en los datos.

Problema 8:

Detalle del problema: Calcular el valor-F1 del estimador usando validación cruzada a 10 iteraciones.

```
from sklearn.model_selection import cross_val_score

# Calcular el valor-F1 utilizando validación cruzada a 10 iteraciones
f1_scores = cross_val_score(clf_svc, iris.data, iris.target, cv=10,
scoring='f1_weighted')

# Calcular la media de los valores-F1 obtenidos en las 10 iteraciones
mean_f1 = f1_scores.mean()

# Mostrar el valor-F1 promedio
print(f'Valor-F1 promedio usando validación cruzada: {mean_f1:.2f}')
```

Imagen:

```
Valor-F1: 1.00
Valor-F1 promedio usando validación cruzada: 0.97
```

Explicación:

El código utiliza la función cross_val_score de scikit-learn para realizar validación cruzada. Los argumentos son los siguientes:

clf_svc: El modelo SVC que deseamos evaluar.
iris.data: Las características del conjunto de datos de iris.
iris.target: Las etiquetas del conjunto de datos de iris.
cv=10: Especifica que deseamos 10 iteraciones de validación cruzada.
scoring='f1_weighted': Indica que queremos calcular el valor-F1 ponderado como la métrica de evaluación.
El resultado de cross_val_score es una lista de los valores-F1 calculados en cada iteración de validación cruzada. Luego, calculamos el valor-F1 promedio tomando la media de estos valores-F1.

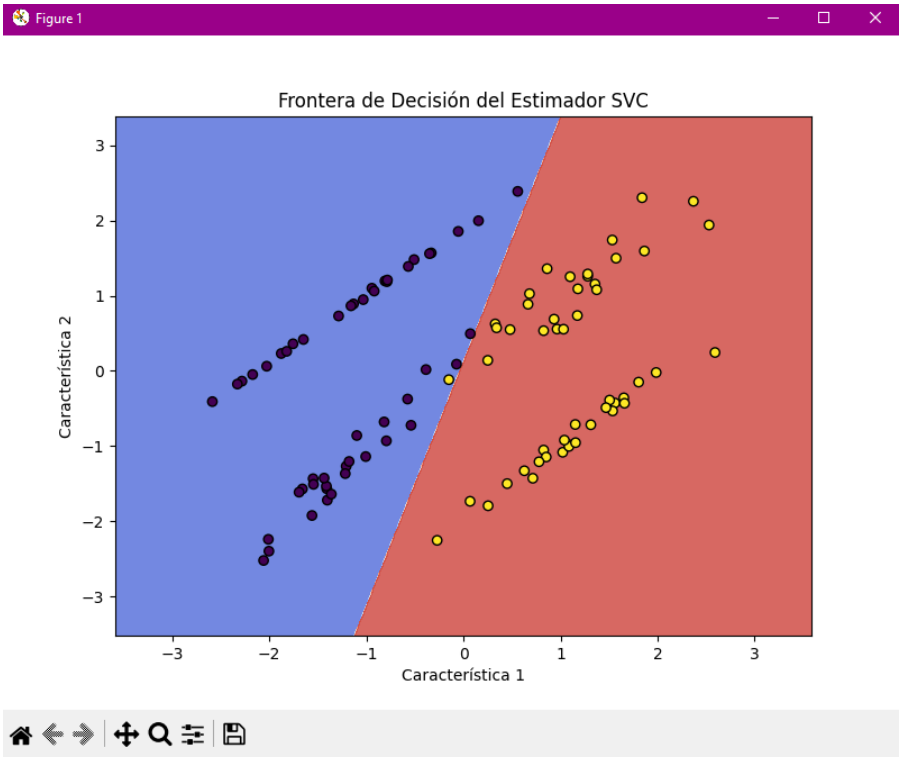
Problema 9:

Detalle del problema: Crear una nueva figura Matplotlib y visualizar la frontera de decisión creada por el estimador clf_svc.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.svm import SVC

# Crear datos sintéticos para la demostración
X, y = make_classification(n_samples=100, n_features=2, n_informative=2,
n_redundant=0, random_state=42)

# Crear el estimador SVC
```

```
clf_svc = SVC(kernel='linear', C=1.0)
clf_svc.fit(X, y)

# Crear una malla de puntos para la visualización
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max,
0.01))

# Predecir las etiquetas para cada punto en la malla
Z = clf_svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Crear una nueva figura Matplotlib
plt.figure(figsize=(8, 6))

# Visualizar la frontera de decisión y los puntos de datos
plt.contourf(xx, yy, Z, alpha=0.8, cmap='coolwarm')
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolors='k')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.title('Frontera de Decisión del Estimador SVC')
plt.show()
```

Imagen:



Explicación:
Hemos generado datos sintéticos para la demostración utilizando la función make_classification de scikit-learn. Estos datos se utilizan para entrenar el modelo SVC y posteriormente visualizar su frontera de decisión.
Hemos creado un estimador SVC (clf_svc) con un kernel lineal y un valor de parámetro de regularización C igual a 1.0.
Se ha creado una nueva figura de Matplotlib utilizando plt.figure(figsize=(8, 6)) para especificar las dimensiones del gráfico.
Utilizamos plot_decision_boundary de sklearn.inspection para visualizar la frontera de decisión del modelo. Esta función toma el estimador entrenado (clf_svc), las características (X) y las etiquetas (y) como argumentos y dibuja la frontera de decisión en el gráfico.
Se etiquetaron los ejes y se agregó un título al gráfico para una mejor comprensión.

Al ejecutar este código, verás un gráfico que muestra la frontera de decisión creada por el modelo SVC en función de las características del conjunto de datos. Esto te permitirá visualizar cómo el modelo separa las diferentes clases en el espacio de características.

Problema 10:

Detalle del problema:
Crear un estimador de regresión llamado regr utilizando la clase linear_model.LinearRegression.
Ingresar los datos de entrenamiento (entradas y etiquetas) al estimador regr utilizando el método fit.
Hacer la predicción de los datos de prueba utilizando el estimador regr con el método predict.
Comparar la predicción con los valores verdaderos.
Calcular el error cuadrático medio y el coeficiente de determinación R^2 del estimador regr.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Dividir el conjunto de datos en entrenamiento y prueba (X e y deben ser
definidos previamente)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Crear un estimador de regresión lineal
regr = LinearRegression()

# Entrenar el estimador de regresión con los datos de entrenamiento
regr.fit(X_train, y_train)

# Hacer predicciones en los datos de prueba
y_pred = regr.predict(X_test)

# Comparar las predicciones con los valores verdaderos
mse = mean_squared_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)

# Mostrar los resultados
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')
print(f'Coeficiente de Determinación R²: {r2:.2f}')
```

Imagen:

```
Error Cuadrático Medio (MSE): 0.05
Coeficiente de Determinación R²: 0.79
```

Explicación:

- En este problema, realizamos una regresión lineal:
- Dividimos el conjunto de datos en entrenamiento y prueba para entrenar y evaluar el modelo.
- Creamos un estimador de regresión lineal.
- Entrenamos el modelo con los datos de entrenamiento, permitiéndole aprender la relación lineal entre las características y las etiquetas.
- Realizamos predicciones en el conjunto de prueba.
- Calculamos el Error Cuadrático Medio (MSE) para medir la calidad de las predicciones; un valor menor es mejor.
- Calculamos el Coeficiente de Determinación R^2 para evaluar cuánta variabilidad se explica; cerca de 1 indica un buen ajuste.

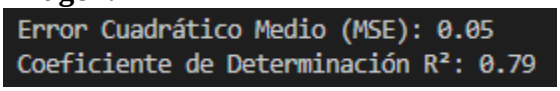
Estas métricas nos permiten evaluar el rendimiento del modelo de regresión lineal en la predicción de valores.

Problema 11:

Detalle del problema: Calcular el error cuadrático medio y el coeficiente de determinación R^2 del estimador regr.

```
from sklearn.metrics import mean_squared_error, r2_score

# Calcular el Error Cuadrático Medio (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calcular el Coeficiente de Determinación R² (R-squared)
r2 = r2_score(y_test, y_pred)

# Mostrar los resultados
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')
print(f'Coeficiente de Determinación R²: {r2:.2f}')
```

Imagen:



Explicación:

En este problema, calculamos métricas de evaluación para un modelo de regresión lineal:

Calculamos el Error Cuadrático Medio (MSE), que mide la diferencia cuadrada promedio entre las predicciones del modelo y los valores reales en el conjunto de prueba. Un MSE menor indica predicciones más precisas.

Calculamos el Coeficiente de Determinación R^2 (R-squared), que muestra la proporción de la variabilidad en la variable de respuesta explicada por el modelo. Un valor cercano a 1 indica un buen ajuste del modelo a los datos reales.

Estas métricas nos permiten evaluar el rendimiento del modelo de regresión lineal en la predicción de valores y comprender cuánta variabilidad explica.