

07/01/2025

Machine Learning I
Rubén Garrido Hidalgo

1.Introducción a Machine Learning

Pregunta teórica: Explica las diferencias entre aprendizaje supervisado y no supervisado, e indica un caso de uso para cada uno.

El aprendizaje supervisado de modelos de inteligencia artificial se caracteriza por entrenar modelos de Machine Learning con un conjunto de datos etiquetados, es decir, cada elemento de un conjunto de datos de entrenamiento se empareja con una etiqueta, la cual corresponde a su salida. A diferencia de un modelo de aprendizaje no supervisado, pues en este tipo de modelos los datos no están clasificados y carecen de etiquetas a diferencia de los datos del aprendizaje anterior, actuando sin orientación sobre los datos.

Los modelos de aprendizaje supervisado aprenden a predecir la salida a partir de los datos de entrada, por ello se emplean en algoritmos de clasificación y regresión. A diferencia de los modelos de aprendizaje no supervisados en los cuales el objetivo es descubrir patrones ocultos o agrupaciones de datos sin haber sido clasificados previamente, es decir aboliendo la intervención humana. Es por ello por lo que este tipo de modelos de aprendizaje no supervisado se emplea en algoritmos de agrupación y asociación.

Caso práctico uso de un modelo de aprendizaje Supervisado:

Podemos emplear un modelo de aprendizaje supervisado como un árbol de decisión para a partir de datos ya definidos como la textura, el peso y el color, clasifiquemos ciertas frutas dadas en los datos.

Caso práctico de aprendizaje no supervisado:

Podemos emplear modelos de aprendizaje no supervisado sobre datos sin etiquetas como dijimos previamente, vemos por ejemplo que si tenemos una empresa en la cual queremos clasificar a los clientes de nuestro negocio (datos de forma aleatoria sin clasificación) en grupos en función de dos datos conocidos como son los ingresos mensuales y los gastos en la tienda. Para ello emplearemos un modelo de aprendizaje no supervisado como puede ser un K-means en este caso.

Pregunta teórica: ¿Qué métricas se pueden utilizar para evaluar un modelo de clasificación y cuáles para uno de regresión?

Las métricas para evaluar un modelo de clasificación son

1. Precisión
2. Matriz de Confusión
3. Área bajo la curva ROC(AUC)

Las métricas para evaluar un modelo de regresión son:

1. Error Absoluto Medio (Mean Absolute Error, MAE):
2. Error Cuadrático Medio (Mean Squared Error, MSE)
3. Raíz del Error Cuadrático Medio (Root Mean Squared Error, RMSE):
4. Coeficiente de Determinación (R^2):
5. Mean Absolute Percentage Error (MAPE)

2. Predictores Lineales

Pregunta teórica: Explica el significado de los coeficientes β_0 y β_1 en un modelo de regresión lineal simple.

$$y = \beta_0 + \beta_1 x + \epsilon$$

El β_0 llamado intercepto, es el término independiente el cual refleja el valor de y cuando $x=0$.

El coeficiente β_1 representa mediante una cantidad numérica la influencia de la variable x_1 sobre la variable dependiente y . Gráficamente es la inclinación de la línea de regresión, la cual marcará como es la pendiente de la recta de regresión.

Si $\beta_1 > 0$ la relación entre x e y es positiva, mientras que si $\beta_1 < 0$ la relación entre ambas es negativa, es decir, cuando una decrece la otra también, se ve con claridad en la ecuación del modelo de regresión.

3. Reducción de dimensionalidad y extracción de características

Pregunta teórica: Define el concepto de reducción de dimensionalidad y cómo este proceso puede mejorar el rendimiento del modelo.

La reducción de la dimensionalidad es el proceso de reducir el número de variables o características en un conjunto de datos simplificando su representación y conservando la mayor parte posible de información relevante para el análisis.

Es un proceso muy útil porque nos permite trabajar con un gran volumen de datos, los cuales están formados por un alto número de variables, es decir, datos de alta dimensionalidad. Permitiendo que gracias a la reducción de dimensionalidad podamos mejorar la eficiencia de nuestros algoritmos de aprendizaje automático, reducir el riesgo de sobreajuste de nuestro modelo y hacer más sencilla la visualización e interpretación de los datos del modelo. También podemos resaltar que la reducción de la dimensionalidad mejora el rendimiento del modelo porque al reducir las variables con las que trabajamos obtenemos un modelo con una menor complejidad computacional y que consigue solventar la ya conocida “maldición de la dimensionalidad”

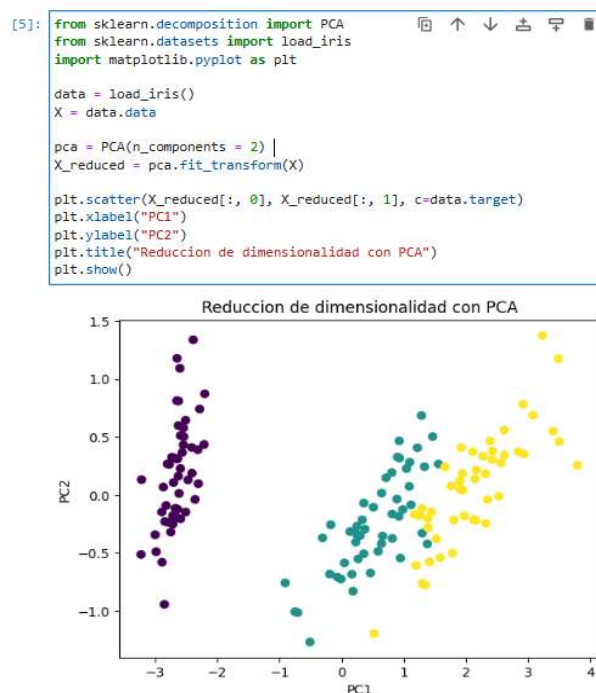
Pregunta teórica: ¿Qué ventajas tiene reducir la dimensionalidad en datos con muchas variables?

Reducir la dimensionalidad de los datos con muchas variables posee diferentes beneficios para nuestro modelo de aprendizaje automático:

- 1- Al reducir la dimensionalidad podemos eliminar variables que no son relevantes, consiguiendo que nuestro modelo de Aprendizaje automático pueda encontrar patrones o relaciones entre las variables de una forma más sencilla, mejorando la precisión del modelo.

- 2- Al trabajar con datos de muchas variables, el modelo tiende a sobre ajustarse a esos datos y a aprender patrones que no permitirán interpretar de manera correcta nuevos datos, mientras que si reducimos la dimensionalidad de los datos evitamos el sobreajuste y conseguimos un modelo más preciso con nuevos datos.
- 3- Al reducir el número de variables, también reducimos el número de cálculos requeridos en las operaciones del modelo, permitiendo un entrenamiento más rápido del modelo y una inferencia más rápida a su vez.
- 4- Reducir la dimensionalidad también nos permite visualizar los datos y comprender de forma más clara la estructura de los datos, sus relaciones y agrupaciones.

Pregunta de programación: Implementa un Análisis de Componentes Principales (PCA) en un conjunto de datos para reducir las dimensiones a 2 y visualiza los datos en el espacio reducido.



Explica los ejes resultantes y su interpretación:

La representación muestra la proyección de los datos del conjunto Iris en las dos primeras componentes principales, calculadas mediante el algoritmo PCA.

```
[4]: #Varianza explicada por cada componente principal
print("Varianza explicada por cada componente principal:")
print(pca.explained_variance_ratio_)

Varianza explicada por cada componente principal:
[0.92461872 0.05306648]
```

Lo que indica que la primera componente principal PC1 explica el 92,4% de la varianza y la segunda componente principal PC2 explica un 5,3% de la varianza. Juntas explican aproximadamente un 97,7% de la información contenida en los datos originales, lo cual nos permite verificar que hemos realizado una reducción de la dimensionalidad satisfactoria hemos conseguido reducir toda la información en dos variables, conservando casi toda la información en su totalidad.

Esto nos permite también decir que las dos primeras componentes principales capturan la mayor parte de la información original y son combinaciones lineales de las variables originales que maximizan la varianza de los datos.

4. Clasificación

Pregunta teórica: Describe el funcionamiento del algoritmo k-vecinos más cercanos (k-NN) y proporciona un caso de uso típico.

El algoritmo de k-vecinos (K-NN) es un algoritmo de aprendizaje supervisado que clasifica una observación en la categoría más común entre sus k vecinos más cercanos, basándose en una medida de distancia como la euclidiana o Manhattan.

A continuación, vamos a describir cómo funciona este algoritmo de clasificación para poder predecir la clase o valor de un punto desconocido:

- 1- Normalizamos los datos que nos proporcionan para evitar algún sobreajuste del modelo o ciertos sesgos que pueden estar presentes en los datos
- 2- Aplicamos a continuación sobre estos datos normalizados el algoritmo de K-NN, donde calculamos la distancia entre un punto de prueba y todos los demás puntos del conjunto de entrenamiento.
- 3- Seleccionamos las k vecinos más cercanos en función de la distancia seleccionada (Euclidiana, Manhattan, Minkowski) con respecto al punto de prueba.
- 4- Asignamos la clase más común al punto de prueba en función de los k vecinos
- 5- Seleccionamos el parámetro k (número de vecinos) teniendo en cuenta que:
 - Si $k=1$ podemos tener un sobreajuste, pues todo depende de un único vecino.
 - Si k posee un valor alto esto suavizará la predicción de la clase, pero también puede ser un problema si k es muy grande, pues podría provocar que el modelo no pueda ajustar bien la clase de un punto al ver que tiene demasiados vecinos de diferentes clases distintas.

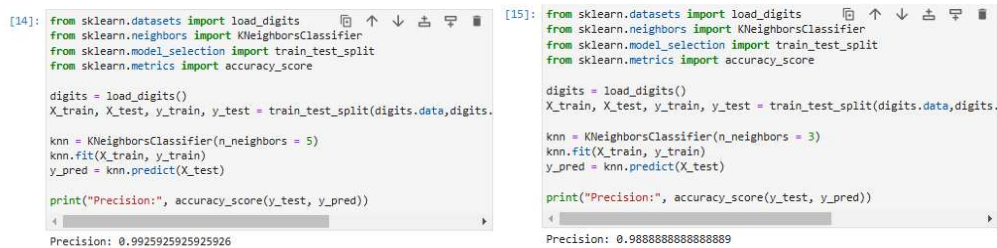
Un caso de uso de este algoritmo se da en el campo de la medicina y la investigación médica, el algoritmo k-NN se puede usar en genética para calcular la probabilidad de ciertas expresiones genéticas. Permitiendo a los médicos predecir la probabilidad de cáncer, ataques cardíacos o cualquier otra afección hereditaria.

Pregunta teórica: Explica el concepto de matriz de confusión y cómo se calcula la precisión

La matriz de confusión es una métrica empleada comúnmente en problemas de clasificación en los árboles de decisión. Permite evaluar el rendimiento de un modelo de clasificación supervisado. Es un elemento el cual consiste en una tabla que resume las predicciones correctas e incorrectas para cada clase. Es una matriz en la cual cada fila representa una clase real y cada columna una predicción de dicha clase.

La **precisión** (Accuracy) se calcula realizando un cociente entre el número de predicciones correctas y el número total de ejemplos.

Pregunta de programación: Implementa un clasificador k-NN en Python para clasificar el conjunto de datos de dígitos en sklearn. Calcula la precisión del modelo y explica su significado con respecto a los parámetros seleccionados para el modelo.



```
[14]: from sklearn.datasets import load_digits
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("Precision:", accuracy_score(y_test, y_pred))

Precision: 0.9925925925925926
```

```
[15]: from sklearn.datasets import load_digits
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target)

knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("Precision:", accuracy_score(y_test, y_pred))

Precision: 0.9888888888888889
```

La precisión del modelo es de 0.992592(99%), si aplicamos el algoritmo de k-NN sobre el conjunto de datos llamados digits, es decir dígitos, para predecir el valor de un dígito en función de sus vecinos vemos que si tomamos como referencia 3 vecinos la precisión de la predicción es de un 98,8%, si aumentamos los vecinos la precisión va aumentando, pues con cinco vecinos la precisión es de 99,2% y con seis vecinos de 99,4, lo que puede llevar aun sobreajuste del modelo. Pero como mencionamos anteriormente, si tomamos más vecinos de los que debemos la precisión baja y se puede producir un subajuste, pues tiene en cuenta tantos vecinos que la predicción del dígito no puede ser correcta, por ejemplo, con 8 vecinos la precisión es de un 98,7% más baja que con 3 vecinos.

Como conclusión podemos decir que el modelo es adecuado para este conjunto de datos de dígitos, tiene clases bien definidas y no es excesivamente grande. Esto lo sabemos porque hemos obtenido una precisión entre un 95-98%.

5. Regresión

Pregunta teórica: Explica el Error Cuadrático Medio (MSE) y su importancia en la evaluación de modelos de regresión.

El Error Cuadrático Medio (MSE) es el promedio de los cuadrados de las diferencias entre los valores reales y los valores predichos de un modelo. El error Cuadrático Medio tiende a penalizar errores grandes, debido a la fórmula que sigue el propio error, pues al elevar las diferencias al cuadrado hace que el error también se eleve al cuadrado. Se define como:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

El Error Cuadrático Medio se emplea en la evaluación de modelos de regresión. Este es una de las métricas empleadas para evaluar el rendimiento de un modelo de regresión.

Para ajustar una regresión lineal o multilineal con n observaciones y p variables independientes vemos que el error cuadrático medio es fundamental para ajustar el modelo, pues actúa como a la función de costo, la cual se busca minimizar para así reducir lo máximo posible el error promedio entre los valores del modelo (valores observado) y los valores predichos, minimizando el error de las predicciones de nuestro modelo de regresión.

Pregunta teórica: ¿Cuándo se ocupa un modelo de regresión? Describe un ejemplo de aplicación de un modelo de regresión.

Un modelo de regresión se emplea cuando queremos predecir una variable continua, es decir, un valor numérico. Un modelo de regresión busca modelar la relación de una variable dependiente con una o más variables independientes, con el fin de predecir valores continuos para la variable dependiente (variable objetivo).

A diferencia de los modelos de clasificación los cuales se basan en realizar predicciones discretas de una variable objetivo (variable dependiente en el modelo de regresión). En vez de valores numéricos se buscan predecir categorías y clases.

Como ejemplo clásico de aplicación de un modelo de regresión podemos intentar predecir el precio de una casa en función de varias variables como son el tamaño de la casa, el número de habitaciones, la antigüedad de la casa y la zona en la que se encuentra.

Tomando el precio como nuestra variable dependiente u objetivo y las demás como variables independientes que tendrán una repercusión sobre nuestro precio representado con un coeficiente beta, el cual indicará la proporción en la que influye sobre el precio cada característica.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \varepsilon$$

También tendremos un error aleatorio (épsilon) el cual regularemos con el MSE descrito anteriormente y nos servirá como criterio para verificar si el modelo es significativo y produce buenas predicciones o si estas no son buenas y por lo tanto debemos de volver a estudiar el modelo.

6. Clustering

Pregunta teórica: Explica el algoritmo K-means y un caso de uso en la vida real.

El K-Means es un algoritmo de agrupamiento basado en particiones que busca dividir el conjunto de datos en k clusters, minimizando la suma de las distancias cuadráticas dentro de cada clúster formado. Vamos a exponer cómo funciona el algoritmo de K-means:

- 1- Inicialización: donde se seleccionan k centroides iniciales aleatoriamente o se sigue algún método heurístico.
- 2- Asignación de clusters: Cada punto de nuestros datos se asigna al clúster más cercano según la distancia que estemos siguiendo, en este caso la distancia euclidiana. Por lo tanto, asignaremos los puntos a un clúster en función del valor de la norma al cuadrado de la diferencia entre el punto y el centroide, eligiendo aquel clúster en el cual dicha diferencia sea mínima.
- 3- Actualización de centroides: cada vez que vamos añadiendo un punto al clúster vamos actualizando los centroides como el promedio de los puntos asignados a cada clúster, cambiando el centroide de cada clúster, en caso de que tengan más de un elemento.

- 4- Convergencia: Se repetirán los pasos 2 y 3 de asignación de clúster y actualización de los centroides hasta que los centroides al ser recalculados no cambien significativamente o se cumpla el criterio de parada de la función de costo:

$$J(\mathcal{C}) = \sum_{i=1}^K \sum_{x \in C_i} \|x - \mu_i\|^2,$$

Es decir, para cierto clúster dicha diferencia es menor que una épsilon mayor que cero

El algoritmo K-means se usa ampliamente hoy en día, uno de sus usos más importantes es su uso en la segmentación de clientes para Marketing. Pues a partir de unos datos permite agrupar a los clientes de una empresa en diferentes clusters o grupos en función de unas características determinadas.

Pregunta teórica: Explica la diferencia entre clustering supervisado y no supervisado.

Las principales diferencias entre el clustering supervisado y el no supervisado son las siguientes

- En el clustering no supervisado los datos vienen sin etiquetas a diferencia de los datos del clustering supervisado donde algunos o todos los datos vienen etiquetados
- El clustering no supervisado, como hemos visto agrupa los datos intentando dividir los datos en clusters donde los elementos tienen ciertas similitudes o son lo más parecidos posibles. Mientras que el clustering supervisado tiene otro enfoque pues al estar todo etiquetado intenta asignar etiquetas o intenta predecir la categoría de los datos en función de lo que ya se conoce
- Por último, debido a este punto anterior los algoritmos de aprendizaje automático empleado son distintos, en el clustering no supervisado se emplearán algoritmos como K-Means o DBSCAN, algoritmos de aprendizaje no supervisado. Mientras que en el clustering supervisado se emplearán algoritmos de aprendizaje supervisado como K-NN.

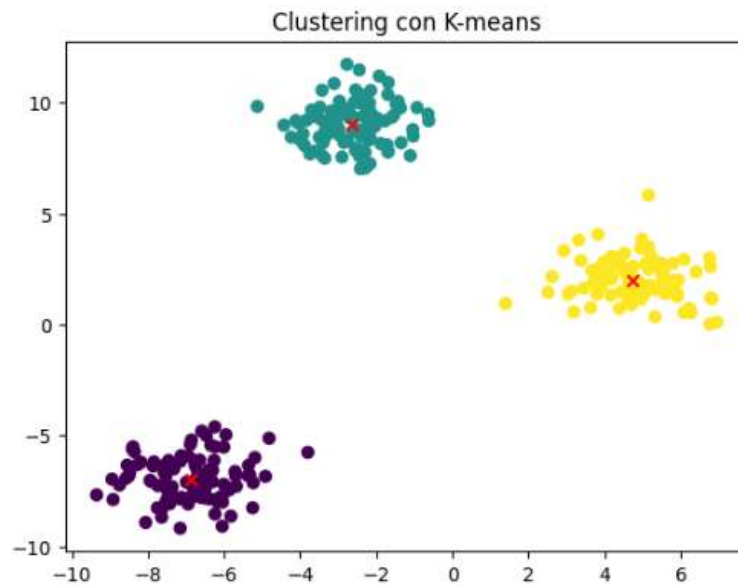
Pregunta de programación: Implementa K-means en Python para agrupar datos en 3 clusters. Muestra los resultados con una gráfica que incluya los centros de cada clúster. ¿Qué conclusiones sobre el modelo de K-means podemos obtener de la gráfica?

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs # no importado la función make_blobs sino no podía cargar

# 300 muestras distribuidas en 3 centros
X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=1.0, random_state=42)

# Crear el modelo K-means
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1],
            color='red', marker='x')
plt.title("Clustering con K-means")
plt.show()
```

De esta gráfica podemos deducir que el algoritmo K-Means está funcionando de forma adecuada. Podemos ver que la elección de tres clusters ha sido muy buena y los datos no se solapan, por lo que es evidente que los datos se clasifican en tres clusters bien diferenciados. Además, otra evidencia del buen funcionamiento del algoritmo es que los centroides se encuentran situados en el centro de los clusters, lo que implica que el algoritmo ha encontrado unos buenos puntos de referencia para cada clúster.

7. Árboles de decisión

Pregunta teórica: Describe el funcionamiento de los árboles de decisión para clasificación.

Los árboles de decisión son algoritmos de aprendizaje supervisado que se utilizan para tareas de clasificación y regresión. La estructura del árbol está compuesta por nodos de decisión, ramas y hojas.

Vamos a ver cómo funcionan los árboles de decisión para clasificación. Como hemos mencionado los árboles son estructuras que constan de tres partes; nodos de decisión, hojas y ramas. Vamos a revisar cómo construimos un árbol de decisión para entender cómo funcionan.

Primero seleccionamos la característica que mejor divide a los datos usando distintas métricas como la ganancia de información, la entropía o el índice de Gini.

Como segundo paso se realiza una división recursiva de los datos en distintos subconjuntos, esto se realiza de manera iterativa.

Por último, se limita la profundidad del árbol o el número mínimo de datos en cada nodo para evitar el sobreajuste.

Por lo tanto, una vez que hemos descrito la construcción del árbol podemos describir cómo funcionaría el mismo para clasificar nuevos ejemplos pues, dado un nuevo dato, este realiza un recorrido descendente desde la raíz del árbol hasta la hoja más profunda que alcance.

El proceso sería el siguiente:

- 1- Comenzamos en la raíz del árbol
- 2- Nos hacemos preguntas para ver si cumple alguna de las características de las hojas de la raíz
- 3- Dependiendo de la respuesta, seguimos a la siguiente rama
- 4- Esto continúa de manera iterativa hasta que se cumple un criterio de parada, como la profundidad o simplemente el algoritmo consigue asignar una nueva clase al nuevo ejemplo

Pregunta teórica: ¿Cómo se puede evitar el sobreajuste en un árbol de decisión?

El sobreajuste es un problema presente en los árboles de decisión. Algunos de los métodos para reducir o evitar el sobreajuste son la poda y la regularización.

La poda puede ser temprana y posterior, la poda temprana se basa en detener la construcción del árbol antes de que alcance su profundidad máxima. Por ejemplo, establecer una profundidad máxima o definir un número mínimo de ejemplos en cada nodo.

Mientras que la poda posterior consiste en construir todo el árbol completo y después eliminar nodos menos significativos para simplificar la estructura.

La regularización es otra técnica mencionada que consiste en agregar restricciones al modelo con el fin de evitar el sobreajuste del árbol de decisión. Algunas de estas técnicas o estrategias son:

- Limitar la profundidad del árbol

- Ajustar el número mínimo de muestras requeridas para dividir un nodo
- Establecer un umbral mínimo para la reducción de la métrica de división

Pregunta de programación: Usa un árbol de decisión para clasificar el conjunto de datos wine. Evalúa la precisión del modelo. ¿Qué conclusiones podrías obtener del valor de la precisión y de la matriz de confusión? ¿Cómo podrías mejorar este modelo?

```
[10]: from sklearn.tree import DecisionTreeClassifier
      from sklearn.datasets import load_wine
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import confusion_matrix
      import numpy as np

      # Cargar el conjunto de datos
      wine = load_wine()

      # Dividir el conjunto de datos en entrenamiento y prueba
      X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3, random_state=42)

      # Entrenar el modelo de árbol de decisión
      tree = DecisionTreeClassifier()
      tree.fit(X_train, y_train)

      # Evaluar precisión en el conjunto de prueba
      print("Precisión en prueba:", tree.score(X_test, y_test))

      # Predicciones en el conjunto de prueba
      y_pred = tree.predict(X_test)

      # Calcular y mostrar la matriz de confusión
      conf_matrix = confusion_matrix(y_test, y_pred)
      print("Matriz de confusión:")
      print(conf_matrix)
```

Precisión en prueba: 0.9629629629629629
 Matriz de confusion:
 [[18 1 0]
 [0 21 0]
 [0 1 13]]

¿Qué conclusiones podrías obtener del valor de la precisión y de la matriz de confusión?

La precisión que hemos obtenido es de aproximadamente un 96,3 %, lo que implica que el modelo clasifica correctamente el 96,3% de los casos en el conjunto de prueba. Este porcentaje nos indica que el modelo tiene buen rendimiento y es capaz de clasificar bien el vino en función de sus características indicadas en el enunciado, con las cuales se ha construido el árbol de decisión.

En cuanto a la matriz de confusión, el resultado ha sido:

```
Matriz de confusion:
[[18  1  0]
 [ 0 21  0]
 [ 0  1 13]]
```

Como indicamos anteriormente en la cuarta actividad las filas representan casos reales y las columnas clases predichas. Por lo tanto, podemos interpretar la matriz de confusión de la siguiente forma;

(Fila 1)

El modelo que estamos implementando (Árbol de decisión) predijo correctamente 18 instancias con clase 0. Es decir, predijo sin equivocarse que 18 de los vinos son del primer tipo. Uno de ellos se clasificó como de segundo tipo por error y ninguno se clasificó erróneamente en el tercer tipo de vino.

(Fila 2)

No se realizaron clasificaciones erróneas en cuanto al primer y tercer tipo de vino. El modelo predijo que 21 vinos eran del segundo tipo

(Fila 3)

En esta última fila el modelo predijo correctamente 13 vinos que eran del tercer tipo, pero hubo un error donde uno de los vinos fue clasificado como de primer tipo siendo este de tercer tipo.

Así analizaríamos la matriz de confusión correspondiente a nuestro problema

¿Cómo podrías mejorar este modelo?

Podríamos mejorar este modelo empleando técnicas vistas anteriormente como por ejemplo emplear una métrica de precisión balanceada sobre el árbol para así evitar que posea clases desbalanceadas:

Ejemplo de aplicación de esto:

```
[11]: from sklearn.tree import DecisionTreeClassifier
      from sklearn.datasets import load_wine
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import confusion_matrix
      import numpy as np

      # Cargar el conjunto de datos
      wine = load_wine()

      # Dividir el conjunto de datos en entrenamiento y prueba
      X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3, random_state=42)

      # Entrenar el modelo de árbol de decisión
      tree = DecisionTreeClassifier(class_weight='balanced')
      tree.fit(X_train, y_train)

      # Evaluar precisión en el conjunto de prueba
      print("Precisión en prueba:", tree.score(X_test, y_test))

      # Predicciones en el conjunto de prueba
      y_pred = tree.predict(X_test)

      # Calcular y mostrar la matriz de confusión
      conf_matrix = confusion_matrix(y_test, y_pred)
      print("Matriz de confusión:")
      print(conf_matrix)
```

Precisión en prueba: 0.9444444444444444
Matriz de confusión:
[[19 0 0]
 [0 21 0]
 [1 2 11]]

También podríamos mejorar el modelo normalizando los datos antes de entrenar nuestro modelo y evitar el sobreajuste limitando algunos hiperparámetros como la profundidad máxima del árbol, el número mínimo de muestras necesarias para dividir un nodo, el número mínimo de muestras requeridas en una hoja, o el número máximo de muestras consideradas en cada división.

Vemos cómo podemos implementar todas estas últimas restricciones sobre nuestro árbol:

max_depth -> Profundidad máxima del árbol.

min_samples_split -> Número mínimo de muestras necesarias para dividir un nodo.

min_samples_leaf -> Número mínimo de muestras requeridas en una hoja.

max_features -> Número máximo de características consideradas en cada división.

Serán las funciones que emplearemos para mejorar el modelo.

```
[12]: from sklearn.tree import DecisionTreeClassifier
      from sklearn.datasets import load_wine
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import confusion_matrix
      import numpy as np

      # Cargar el conjunto de datos
      wine = load_wine()

      # Dividir el conjunto de datos en entrenamiento y prueba
      X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3, random_state=42)

      # Entrenar el modelo de árbol de decisión
      tree = DecisionTreeClassifier(max_depth=5, min_samples_split=4, min_samples_leaf=2, random_state=42)
      tree.fit(X_train, y_train)

      # Evaluar precisión en el conjunto de prueba
      print("Precisión en prueba:", tree.score(X_test, y_test))

      # Predicciones en el conjunto de prueba
      y_pred = tree.predict(X_test)

      # Calcular y mostrar la matriz de confusión
      conf_matrix = confusion_matrix(y_test, y_pred)
      print("Matriz de confusión:")
      print(conf_matrix)

      Precision en prueba: 0.9814814814814815
      Matriz de confusion:
      [[19  0  0]
       [ 0 21  0]
       [ 1  0 13]]
```

Aumentando la precisión de la prueba y obteniendo una matriz de confusión muy buena y mejor que la que hemos obtenido anteriormente.

8. Métodos por conjuntos

Pregunta teórica: Compara los métodos de Bagging y Boosting.

El Boosting es un método por conjuntos que combina modelos de forma secuencial, donde cada modelo intenta corregir los errores cometidos por los modelos anteriores. Mientras que el Bagging o Bootstrap Aggregating es un método por conjuntos que construye múltiples modelos a partir de diferentes subconjuntos de datos generados mediante muestreo por reemplazo.

En el método de Bagging las predicciones finales se obtienen combinando las predicciones de los modelos individuales mientras que en el Boosting los modelos posteriores se enfocan más en las observaciones mal clasificadas o con mayor error.

Pregunta teórica: ¿Por qué los métodos por conjuntos suelen tener mejor rendimiento que los modelos individuales?

Los métodos por conjunto suelen tener mejor rendimiento que los modelos individuales porque como su propio nombre indican combinan múltiples predicciones, lo que hace posible que estos métodos por conjuntos sean capaces de disminuir el error de sesgo y el error de varianza. Esto se produce por dos motivos clave, gracias a la disminución de errores sistemáticos se produce una reducción del error de sesgo y al suavizar errores específicos de cada modelo se puede disminuir el error de la varianza.

También reducen el sobreajuste del modelo final que se emplea gracias a la capacidad de los modelos de votar entre modelos lo cual permite mejorar la estabilidad y escoger aquel que sea menos propenso al sobreajuste.

Pregunta de programación: comparar el desempeño de un Árbol de Decisión y un modelo de Random Forest en un conjunto de datos artificial generado con dos características relevantes. Deberías entrenar ambos modelos, calcular la precisión de cada uno en los datos de prueba y analizar las diferencias. A partir de los resultados, reflexiona sobre las diferencias entre las fronteras de decisión en las gráficas las precisiones obtenidas, destacando las ventajas y desventajas de cada enfoque. Finalmente, concluye explicando en que situaciones sería más adecuado utilizar un modelo de Random Forest frente a un Árbol de Decisión individual.

```
[7]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generar datos artificiales
X, y = make_classification(n_samples=300, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=42)

# Dividir datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Entrenar un nico rbol de decisi n
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# Entrenar un modelo de Random Forest
forest = RandomForestClassifier(random_state=42)
forest.fit(X_train, y_train)

# Predicciones
y_pred_tree = tree.predict(X_test)
y_pred_forest = forest.predict(X_test)

# Calcular precisi
accuracy_tree = accuracy_score(y_test, y_pred_tree)
accuracy_forest = accuracy_score(y_test, y_pred_forest)
print(f"Precisi n del rbol de Decisi n: {accuracy_tree:.2f}")
print(f"Precisi n del Random Forest: {accuracy_forest:.2f}")

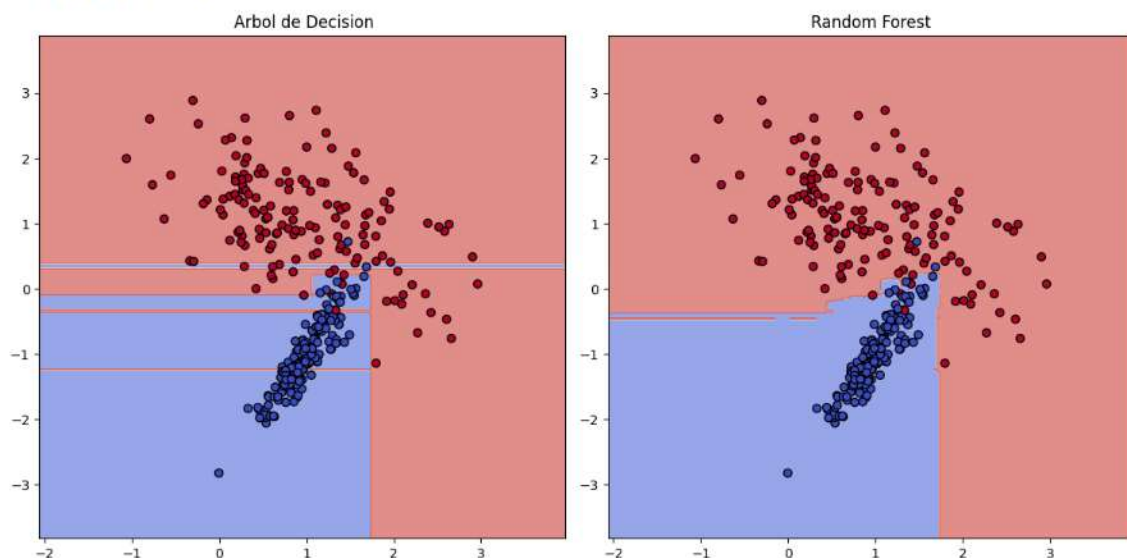
# Visualizar las fronteras de decisi n
def plot_decision_boundary(clf, X, y, ax, title):
    x_min, x_max = X[:, 0].min()- 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min()- 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.6, cmap='coolwarm')
    ax.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap='coolwarm')
    ax.set_title(title)

# Crear las gr ficas
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
plot_decision_boundary(tree, X, y, axes[0], title="Arbol de Decision")
plot_decision_boundary(forest, X, y, axes[1], title="Random Forest")
plt.tight_layout()
plt.savefig('MC.png')
```

Precisi n del rbol de Decisi n: 0.89
Precisi n del Random Forest: 0.94



A partir de los resultados, reflexiona sobre las diferencias entre las fronteras de decisi3n en las gráficas las precisiones obtenidas, destacando las ventajas y desventajas de cada enfoque.

En cuanto a las fronteras de decisi3n:

el árbol de decisión genera fronteras más rígidas y estrictas, las cuales vemos que están escalonadas lo cual puede ajustarse demasiado a los datos y puede causar un sobreajuste del modelo a los datos.

Mientras que el Random Forest al combinar múltiples árboles produce fronteras de decisión más flexibles y generalizadas, lo cual permite capturar patrones complejos y también permite generalizar de forma más sencilla sin caer en el sobreajuste.

En cuanto a la precisión:

El Random Forest es evidente que supera en cuanto hablamos de precisión al Árbol de Decisión, ya que vemos en la imagen que Random Forest ha reducido la varianza de los datos y ha conseguido capturar más datos dentro de la frontera de decisión.

Ventajas y desventajas de ambos modelos:

La ventaja principal del Árbol de decisión es que es más interpretable y más rápido de entrenar, mientras que su mayor desventaja es el mayor riesgo de sufrir un overfitting.

La ventaja principal de un modelo Random Forest es que es más robusto frente a ruido y menos propenso al sobreajuste, mientras que su mayor desventaja es que es menos interpretable y más costoso computacionalmente.

Como conclusión debemos destacar que emplear un Random Forest frente a un Árbol de Decisión es mejor en situaciones donde:

1. Trabajamos con datos complejos
2. El modelo es propenso a sobre ajustarse
3. Trabajamos con grandes volúmenes de datos
4. Para tareas de clasificación y regresión compleja

9. Kernel Ridge

Pregunta teórica: ¿Cómo combina el Kernel Ridge la regularización y los métodos de Kernel?

Como sabemos la Kernel Ridge Regression es una técnica de regularización para modelos de regresión lineal que penaliza la magnitud de los coeficientes del modelo, reduciendo el riesgo de sobreajuste (overfitting) al ruido en los datos.

Para combinar la regularización y los métodos de Kernel en la Kernel Ridge Regression se emplea el truco del Kernel. Para incorporar este truco sobre la KRR, primero debemos representar los w pesos en la forma dual:

$$\mathbf{w} = \Phi^T \mathbf{a},$$

W será el producto entre un vector de coeficientes duales a y una matriz de características transformada. Al sustituir esta representación en el modelo de Ridge Regression, la función de pérdida se reescribe como:

$$L(\mathbf{a}) = \frac{1}{2} \|\mathbf{y} - \mathbf{K}\mathbf{a}\|^2 + \frac{\lambda}{2} \mathbf{a}^T \mathbf{K}\mathbf{a},$$

Donde K es la matriz de Gram.

También se emplea la regularización para así evitar el sobreajuste del modelo en espacios de alta dimensión. Esto se hace introduciendo un término de regularización.

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (y_i - \phi(\mathbf{x}_i)^T \mathbf{w})^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

En el Kernel Ridge Regression, la regularización y el kernel se combinan de la siguiente forma:

- Kernel: permite modelar relaciones no lineales entre las características mediante la transformación implícita de los datos al espacio de características superior.
- Regularización: controla la magnitud de los coeficientes de la combinación de características en ese espacio transformado, evitando sobreajustes al controlar la complejidad del modelo.

Es la combinación de ambos la cual permite al modelo ser capaz de aprender una representación no lineal mientras mantiene una estructura regularizada que previene el sobreajuste. Es decir, el Kernel nos permite capturar relaciones complejas de los datos, mientras que la regularización controla la magnitud de la solución. Favoreciendo el uso de modelos más simples con mayor capacidad de generalización.

Pregunta teórica: ¿Cuáles son las ventajas de utilizar un Kernel en problemas no lineales?

Las principales ventajas son la eficiencia, pues permite trabajar con espacios de alta dimensionalidad sin necesidad de calcular explícitamente las transformaciones. La flexibilidad, porque permite trabajar con cualquier kernel válido, lo que permite modelar relaciones no lineales complejas. Y, por último, la última ventaja que quiero resaltar es la generalización, pues utiliza una representación matemática bien definida y regularizada.

Pregunta de programación: Implementa Kernel Ridge y evalúa su MSE.

```
[10]: from sklearn.kernel_ridge import KernelRidge
      from sklearn.metrics import mean_squared_error

      kr = KernelRidge(alpha=1.0, kernel='rbf')
      kr.fit(X_train, y_train)
      y_pred = kr.predict(X_test)

      print("MSE Kernel Ridge:", mean_squared_error(y_test, y_pred))

MSE Kernel Ridge: 0.035236107577580285
```

El error cuadrático medio del modelo es 0.0352 lo cual nos indica que las predicciones realizadas por el modelo se encuentran muy cerca de los valores reales. Esto nos indica que el modelo puede capturar bien las relaciones en los datos de entrenamiento empleados.