Universidad Tecnológica De Querétaro

2da Evaluación

Diego Ulises Mireles Marcial

martes, 22 de agosto de 2023

UTEQ, Av. Pie de la Cuesta 2501, Nacional, 76148 Santiago de Querétaro, Qro.

Grupo: IDSG05

Contenido

SA

SA-1 Análisis supervisado (Beisbol)

1. Basado en el conjunto de datos "beisbol.csv", implemente el algoritmo de regresión de su preferencia y entregue un reporte que incluya:

Justificación del algoritmo.

La elección de emplear el método de regresión Ridge se fundamenta en su aplicabilidad al problema de predecir las puntuaciones de carreras en el contexto del béisbol. Ridge es una técnica dentro de la regresión lineal que ayuda a suavizar los resultados de la predicción al reducir la influencia excesiva de ciertas características en el modelo final. Al introducir un término de ajuste controlado por el parámetro alfa, Ridge logra mantener una balanceada relación entre la precisión en los datos de entrenamiento y la capacidad de generalizar a nuevos datos.

Descripción del diseño del modelo

El diseño del modelo implica los siguientes pasos:

- Se cargan los datos del archivo CSV en un DataFrame y se importan librerias.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
```

[2]:
```
# Cargar el conjunto de datos
beisbol_data = pd.read_csv(r'C:\Users\Diego\OneDrive\Escritorio\DatosEvaluacion\beisbol.csv')
```

- Se realiza el preprocesamiento de datos: Se codifica la columna categórica 'equipos' utilizando one-hot encoding para convertirla en variables numéricas.

[3]:
```
# Codificar la columna 'equipos' usando one-hot encoding
beisbol_data = pd.get_dummies(beisbol_data, columns=['equipos'], drop_first=True)
```

- Se dividen los datos en características (X) y variable objetivo (y).

[4]:
```
# Dividir el conjunto de datos en características (X) y variable objetivo (y)
X = beisbol_data.drop('runs', axis=1)
y = beisbol_data['runs']
```

- Se divide el conjunto de datos en conjuntos de entrenamiento y prueba utilizando la función train_test_split de Scikit-learn.

```
[5]: # Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Se crea una instancia del modelo de regresión lineal (Ridge) y se entrena en los datos de entrenamiento utilizando el método fit.

```
[6]: # Crear y entrenar el modelo de regresión Ridge
ridge_model = Ridge()
ridge_model.fit(X_train, y_train)
```

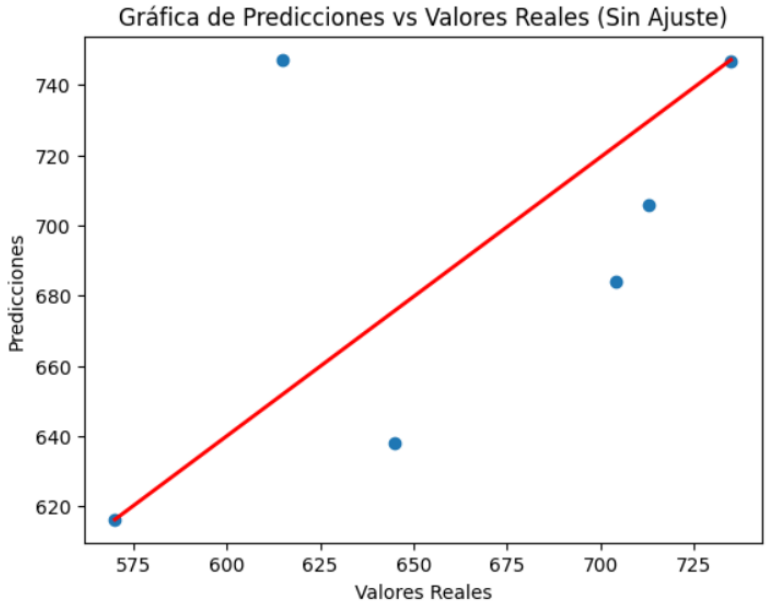
```
[6]: ▾ Ridge
     Ridge()
```

```
[7]: # Realizar predicciones en el conjunto de prueba
y_pred = ridge_model.predict(X_test)
```

```
[8]: # Evaluar el modelo sin ajuste de hiperparámetros
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

- Se grafican los resultados del modelo

```
[9]: # Crear una gráfica de dispersión y línea de regresión para el modelo sin ajuste de hiperparámetros
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_pred), max(y_pred)], color='red', linewidth=2)
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Gráfica de Predicciones vs Valores Reales (Sin Ajuste)')
plt.show()
```



Evaluación y optimización del modelo.

Después de entrenar el modelo, se realizan predicciones en el conjunto de prueba. Luego, se calcula el error cuadrático medio (MSE) y el coeficiente de determinación (R^2) para evaluar el rendimiento del modelo. Estas métricas nos ayudan a entender qué tan bien se ajusta el modelo a los datos reales y cómo de bien puede predecir nuevas observaciones.

[23]:
```
# Imprimir resultados del modelo sin ajuste de hiperparámetros
print("Error cuadrático medio (MSE):", mse)
print("Coeficiente de determinación (R²):", r2)
```

```
Error cuadrático medio (MSE): 3372.006810536046
Coeficiente de determinación (R²): 0.019733799708504418
```

Hiperparámetros y optimización

- Definimos los valores para los hiperparámetros

[11]:
```
# Definir los valores de los hiperparámetros que deseas probar
param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100],  # Valores de regularización (alpha)
    'fit_intercept': [True, False]
}
```

- Se escalan los datos para Ridge, creamos el modelo de nuevo y un objeto GridSearchCV con el modelo, la cuadrícula de hiperparámetros y la métrica a optimizar, seguido, ajustamos el GridSearchCv a los datos escalados

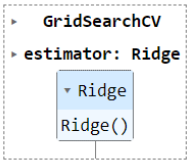
[12]:
```
# Escalar los datos para Ridge
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[13]:
```
# Crear el modelo de regresión Ridge
ridge_model = Ridge()
```

[14]:
```
# Crear un objeto GridSearchCV con el modelo, la cuadrícula de hiperparámetros y la métrica a optimizar (por ejemplo, MSE)
grid_search = GridSearchCV(ridge_model, param_grid, scoring='neg_mean_squared_error', cv=5)
```

[15]:
```
# Ajustar el objeto GridSearchCV a tus datos de entrenamiento escalados
grid_search.fit(X_train_scaled, y_train)
```

[15]:



- Obtenemos los mejores hiperparámetros y puntuación

[16]:
```
# Obtener los mejores hiperparámetros y la mejor puntuación
best_params = grid_search.best_params_
best_score = -grid_search.best_score_
```

- Se crea un nuevo modelo Ridge con estos hiperparámetros, se realizan las predicciones y evaluamos el modelo

[17]:
```
# Crear un nuevo modelo de regresión Ridge con los mejores hiperparámetros encontrados
best_ridge_model = Ridge(**best_params)
```

[18]:
```
# Entrenar el modelo con los datos de entrenamiento escalados
best_ridge_model.fit(X_train_scaled, y_train)
```

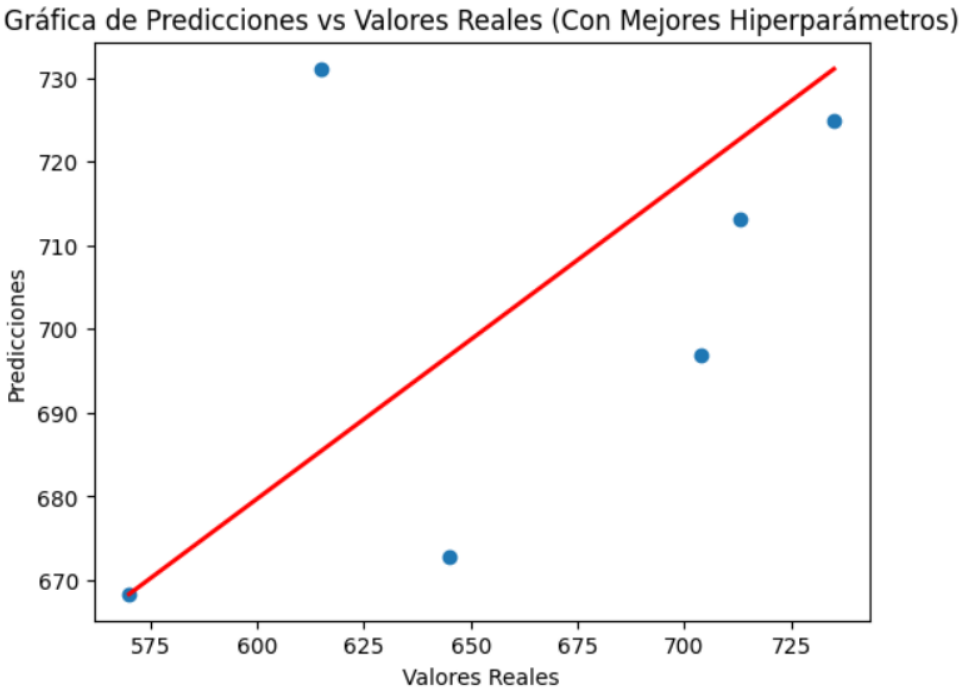
[18]:
```
Ridge
Ridge(alpha=10)
```

[19]:
```
# Realizar predicciones en el conjunto de prueba escalado
y_pred_best = best_ridge_model.predict(X_test_scaled)
```

[20]:
```
# Evaluar el modelo con los mejores hiperparámetros
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
```

- Se crea la gráfica de dispersión y línea de regresión para modelo con mejores hiperparámetros:

[21]:
```
# Crear una gráfica de dispersión y línea de regresión para el modelo con mejores hiperparámetros
plt.scatter(y_test, y_pred_best)
plt.plot([min(y_test), max(y_test)], [min(y_pred_best), max(y_pred_best)], color='red', linewidth=2)
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Gráfica de Predicciones vs Valores Reales (Con Mejores Hiperparámetros)')
plt.show()
```



- Finalmente imprimimos nuevamente los resultados y podemos observar que estos se han ajustado

```
[25]: # Imprimir resultados del modelo con mejores hiperparámetros
print("Error cuadrático medio (MSE) con mejores hiperparámetros:", mse_best)
print("Coeficiente de determinación (R²) con mejores hiperparámetros:", r2_best)
```

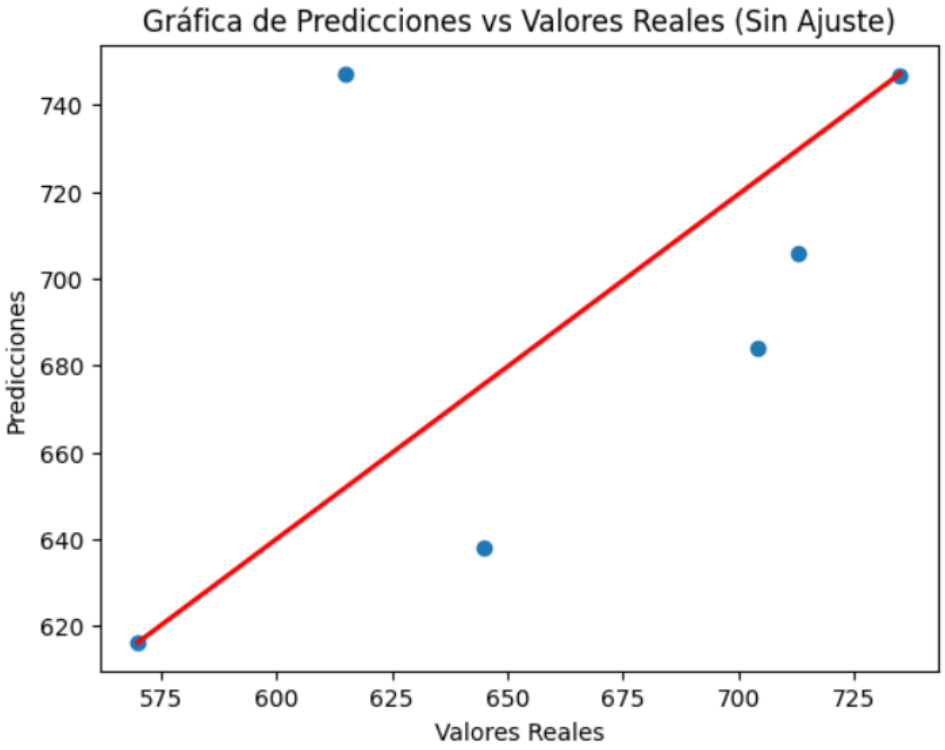
```
Error cuadrático medio (MSE) con mejores hiperparámetros: 4010.3244795703845
Coeficiente de determinación (R²) con mejores hiperparámetros: -0.16582965587174825
```

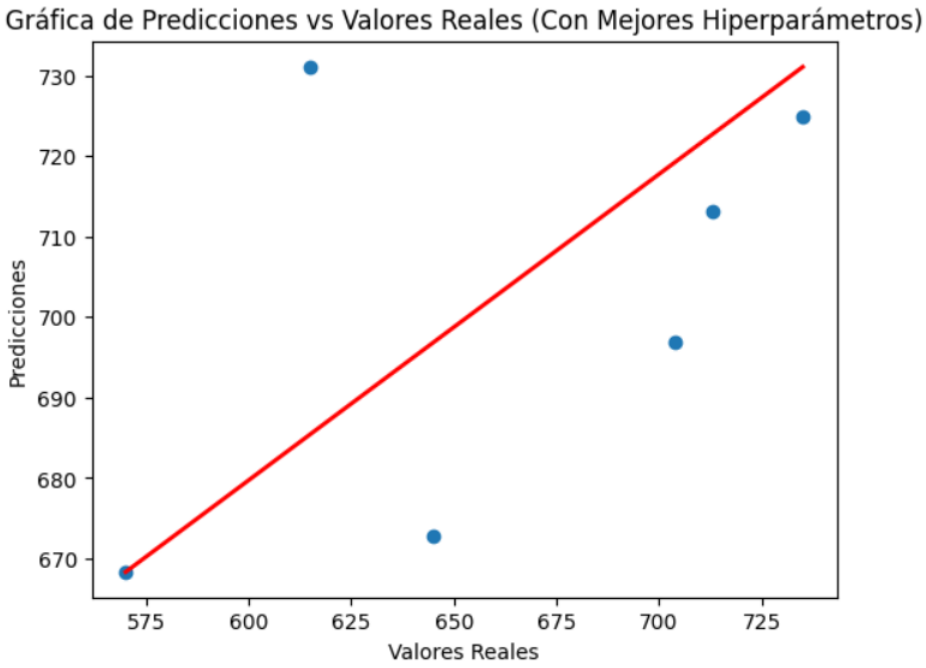
Enlace hacia un repositorio que contenga el modelo obtenido.

Para poder acceder a los datos y resultados, visitar el siguiente repositorio:

https://github.com/DiegoUlisesMM/Eva2BD

Gráfica personalizada e interpretación de resultados.





La grafica representa visualmente la relación entre las puntuaciones predichas por el modelo de regresión y los valores reales de carreras anotadas en los partidos de béisbol. Cada punto en la gráfica representa un equipo y su respectiva puntuación real en el eje horizontal, mientras que en el eje vertical se encuentran las puntuaciones predichas por el modelo. Una dispersión cercana a la línea diagonal indica una predicción precisa, donde las puntuaciones predichas se alinean estrechamente con las puntuaciones reales.

La gráfica muestra cómo las predicciones del mejor modelo Ridge se comparan con los valores reales. Si los puntos están cerca de la línea roja, las predicciones son cercanas a los valores reales. La línea roja representa la tendencia de las predicciones.

SA-2 Análisis supervisado (diabetes_indiana)

2. Basado en el conjunto de datos "diabetes_indiana.csv" implemente el algoritmo de clasificación de su preferencia y entregue un reporte que incluya:

Justificación del algoritmo.

Se eligió utilizar el algoritmo RandomForestClassifier para abordar el problema de clasificación en el conjunto de datos de diabetes de Indiana. La justificación detrás de esta elección radica en las siguientes razones:

- Flexibilidad y Robustez: RandomForestClassifier es un método de aprendizaje automático versátil que puede manejar tanto problemas de clasificación como de regresión. Son resistentes al sobreajuste y funcionan bien en una variedad de situaciones.
- Manejo de Características: Es capaz de manejar múltiples características y son adecuados para conjuntos de datos con características numéricas y categóricas.
- Reducción de Variabilidad: Al promediar las predicciones de varios árboles de decisión, los bosques aleatorios tienden a reducir la variabilidad y el ruido en las predicciones.

Descripción del diseño del modelo.

El diseño del modelo se realizó siguiendo los siguientes pasos:

- Carga de Datos: Se cargó el conjunto de datos "diabetes_indiana.csv" que contiene información sobre pacientes diabéticos.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
# Cargar los conjuntos de datos
diabetes_data = pd.read_csv(r'C:\Users\Diego\OneDrive\Escritorio\DatosEvaluacion\diabetes_indiana.csv')
```

- Preprocesamiento de Datos: Se eliminó la columna "Unnamed: 0" que parece ser un índice no necesario. Luego, se dividieron los datos en características (variables independientes) y la variable objetivo (variable dependiente).

[3]:
```
# Dividir los datos en características (X) y variable objetivo (y)
X = diabetes_data.drop(['Unnamed: 0', '8'], axis=1)
y = diabetes_data['8']
```

- División en Conjuntos de Entrenamiento y Prueba: Los datos se dividieron en conjuntos de entrenamiento (80%) y prueba (20%) utilizando la función train_test_split.

[4]:
```
# Dividir el conjunto de datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Creación y Entrenamiento del Modelo: Se creó un modelo RandomForestClassifier con una semilla aleatoria de 42 y se entrenó utilizando el conjunto de entrenamiento.

[5]:
```
# Crear y entrenar el modelo RandomForest
random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(X_train, y_train)
```

[5]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

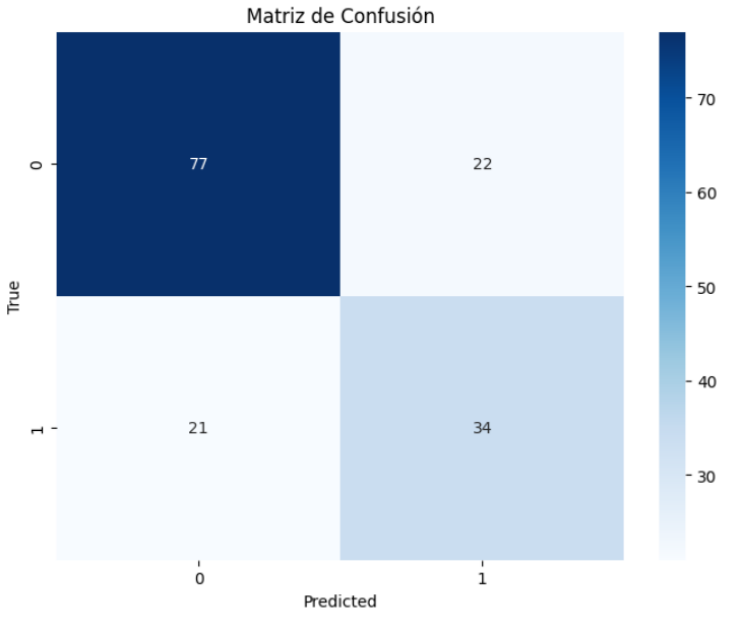
- Evaluación del Modelo: Se realizaron predicciones en el conjunto de prueba y se generó un informe de clasificación que incluye métricas como precisión, recall, F1-score y soporte.

[6]:
```
# Evaluar el modelo en el conjunto de prueba
y_pred = random_forest_model.predict(X_test)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

[7]:
```
# Mostrar el informe de clasificación
print("Informe de Clasificación:")
print(classification_rep)
```

```
Informe de Clasificación:
              precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

[8]:
```
# Graficar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Matriz de Confusión')
plt.show()
```



Evaluación y optimización del modelo.

Se realiza una optimización adicional de los hiperparámetros del modelo RandomForest para mejorar su rendimiento. El código realiza una evaluación cuantitativa del modelo RandomForest utilizando métricas de evaluación específicas y realiza ajustes de los hiperparámetros con el objetivo de optimizar el rendimiento del modelo en la tarea de clasificación de la diabetes.

Los hiperparámetros que se ajustan incluyen el número de estimadores (n_estimators), la profundidad máxima de los árboles (max_depth), el número mínimo de muestras requeridas para dividir un nodo (min_samples_split) y el número mínimo de muestras requeridas en un nodo hoja (min_samples_leaf).

Estos hiperparámetros afectan la complejidad y la capacidad de generalización del modelo. Al ajustarlos, se busca encontrar un equilibrio entre el ajuste excesivo (overfitting) y la falta de ajuste (underfitting), mejorando así el rendimiento en datos no vistos.

[9]:
```
# Crear y entrenar el modelo RandomForest con ajustes de hiperparámetros
random_forest_model = RandomForestClassifier(
    n_estimators=300,
    max_depth=20,
    min_samples_split=5,
    min_samples_leaf=2,
    max_features='sqrt',
    bootstrap=True,
    random_state=42
)
random_forest_model.fit(X_train, y_train)
```

[9]:
```
RandomForestClassifier(max_depth=20, min_samples_leaf=2, min_samples_split=5,
                       n_estimators=300, random_state=42)
```

[10]:
```
# Evaluar el modelo en el conjunto de prueba
y_pred = random_forest_model.predict(X_test)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

[11]:
```
# Imprimir el reporte de clasificación y la matriz de confusión
print("Reporte de Clasificación:\n", classification_rep)
print("Matriz de Confusión:\n", conf_matrix)
```

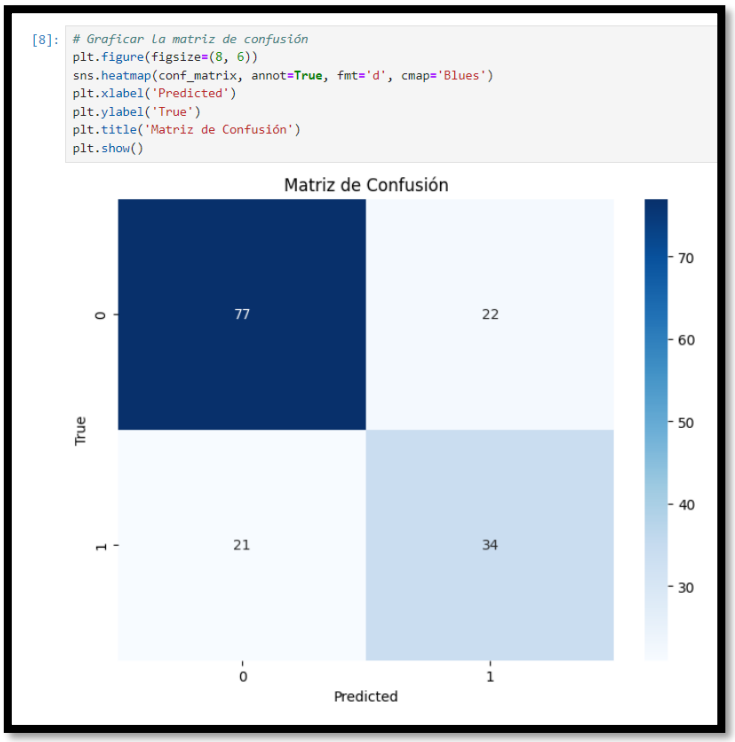
```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.82      0.78      0.80        99
           1       0.63      0.69      0.66        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154

Matriz de Confusión:
 [[77 22]
 [17 38]]
```

Enlace hacia un repositorio que contenga el modelo obtenido.

Para poder acceder a los datos y resultados, visitar el siguiente repositorio:

https://github.com/DiegoUlisesMM/Eva2BD

Gráfica personalizada e interpretación de resultados.



La gráfica de la matriz de confusión generada a partir de los archivos de diabetes de Indiana proporciona una representación visual de cómo el modelo de clasificación está realizando las predicciones en términos de verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN). Cada una de las celdas de la matriz muestra la cantidad de instancias que caen en esa categoría.

En esta gráfica, si enfocamos en la diagonal principal (de la esquina superior izquierda a la inferior derecha), podemos observar que los verdaderos positivos y verdaderos negativos son mayores en comparación con los falsos positivos y falsos negativos. Esto indica que el modelo está acertando en la clasificación de las instancias en ambas clases (diabéticas y no diabéticas) en general.

La gráfica también puede ayudarnos a identificar en qué clase el modelo tiende a equivocarse más. Si los falsos positivos son significativamente más altos que los falsos negativos, podría indicar que el modelo está siendo más propenso a clasificar erróneamente instancias como positivas cuando en realidad son negativas. Del mismo modo, si los falsos negativos son mucho mayores que los falsos positivos, podría indicar que el modelo está perdiendo instancias que son positivas, pero se clasifican como negativas.

SA-3 Análisis no supervisado (Samsung)

3. Basado en el conjunto de datos "samsung.csv" implemente el algoritmo de agrupación de su preferencia y entregue un reporte que incluya:

Justificación del algoritmo.

El algoritmo de agrupación K-Means ha sido seleccionado para este análisis debido a su simplicidad y eficacia en la agrupación de datos numéricos. K-Means es ampliamente utilizado en problemas de segmentación de datos y es capaz de agrupar observaciones similares en clusters basados en medidas de distancia. A pesar de que no está diseñado específicamente para series temporales, podemos adaptarlo para analizar cómo se agrupan los datos de precios de cierre y volúmenes de acciones de Samsung en un espacio bidimensional.

Descripción del diseño del modelo.

- Carga de Datos: Se cargaron los datos de precios de cierre y volúmenes de acciones de Samsung desde el archivo "samsung.csv".

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import os
os.environ['LOKY_MAX_CPU_COUNT'] = '4'
```

[2]:
```
# Cargar el conjunto de datos
data_path = r'C:\Users\Diego\OneDrive\Escritorio\DatosEvaluacion\samsung.csv'
samsungData = pd.read_csv(data_path)
```

- Selección de Características: Se seleccionaron las características "Close" y "Volume" para el análisis de agrupación.

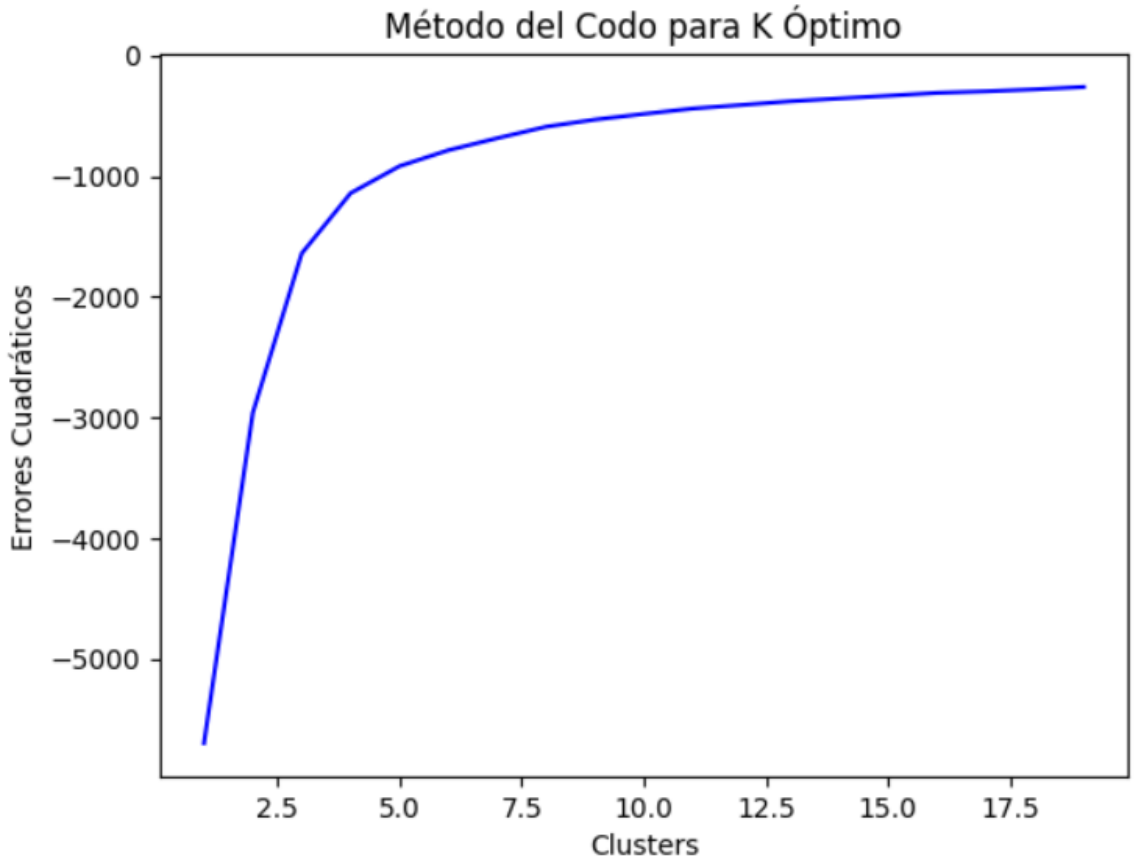
[3]:
```
# Seleccionar las características para el análisis de agrupación
features = ['Close', 'Volume']
data = samsungData[features]
```

- Normalización de Datos: Los datos fueron normalizados utilizando la clase StandardScaler para garantizar que las características tengan la misma escala.

[4]:
```
# Normalizar los datos
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

- Elección del Número de Clusters (K): Se eligió un número de clusters igual a 3, pero esta elección puede variar dependiendo de los objetivos del análisis.

[14]:
```
# Determinar el número óptimo de clusters usando el método del Codo
N = range(1, 20)
kmeans = [KMeans(n_clusters=i) for i in N]
score = [kmeans[i].fit(scaled_data).score(scaled_data) for i in range(len(kmeans))]

# Graficar la curva del Codo
plt.xlabel('Clusters')
plt.ylabel('Errores Cuadráticos')
plt.title('Método del Codo para K Óptimo')
plt.plot(N, score, color='blue')
plt.show()
```



Con estos resultados podemos intuir que el valor ideal para K (clusters) es entre 7.5 y 12.5, por lo tanto, usaremos 10.0.

Mediante K-Means, llevamos a cabo una predicción sobre a qué grupo pertenece cada punto de datos. Cada punto es asignado al grupo cuyo centroide está más cercano. Las predicciones resultantes se almacenan en una nueva columna en nuestros datos.

[23]:
```
#Predicción de clusteres
labels = kmeans.predict(scaled_data)
samsungData['label'] = labels
samsungData
```

[23]:

	Date	Close	Volume	cluster	label
0	02/01/2008	10880	18047200	3	3
1	03/01/2008	10920	19346500	3	3
2	04/01/2008	10780	17997350	3	3
3	07/01/2008	10380	39787200	4	4
4	08/01/2008	10320	24783700	3	3
...	...	...	...	...	...
2845	24/06/2019	45500	6085066	2	2
2846	25/06/2019	45600	7076774	2	2
2847	26/06/2019	45700	9226097	2	2
2848	27/06/2019	46500	12603534	2	2
2849	28/06/2019	47000	12949231	2	2

2850 rows × 5 columns

Implementamos el algoritmo K-Means con nuestros datos y el número de grupos establecidos. A través de iteraciones, el algoritmo ajusta los centroides para minimizar la distancia entre los puntos de datos y los centroides de los grupos.

Integramos estas predicciones de grupos en nuestros datos originales. Esto nos permite ver en qué grupo K-Means ha colocado cada punto de datos en función de sus características.

```
# Aplicar K-Means
kmeans = KMeans(n_clusters=num_clusters, n_init=10, random_state=42)
kmeans.fit(scaled_data)
```

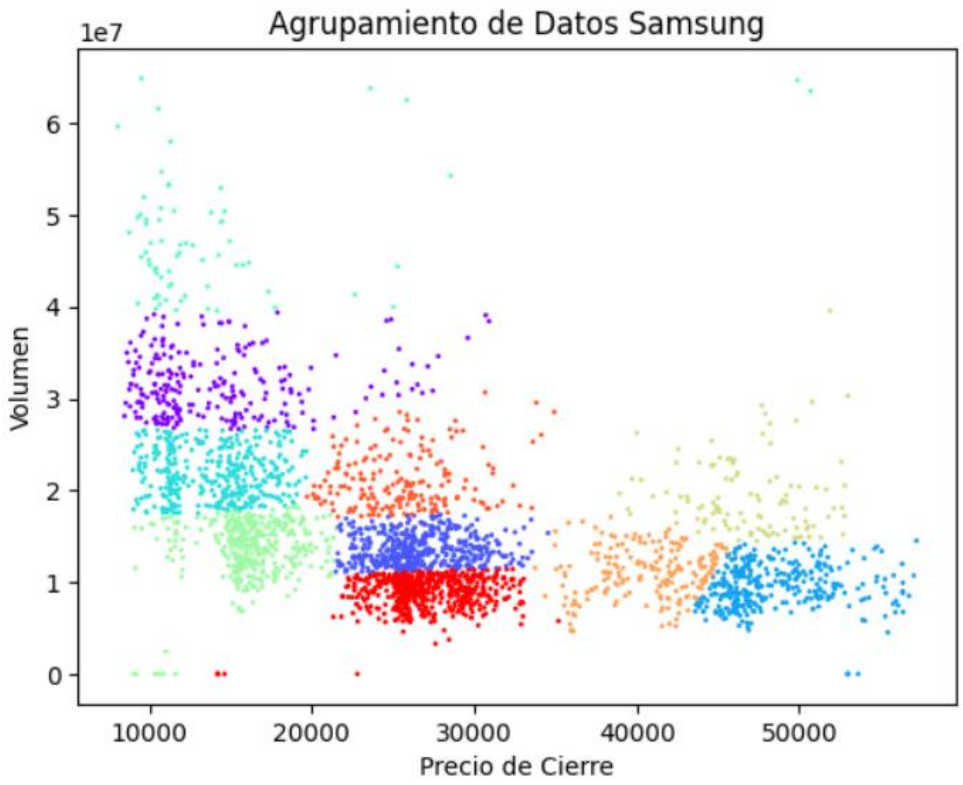
```
KMeans
KMeans(n_clusters=10, n_init=10, random_state=42)
```

[25]:
```
# Agregar las etiquetas de los clusters al DataFrame
samsungData['cluster'] = kmeans.labels_
```

Finalmente, creamos una representación visual utilizando un gráfico de dispersión. Cada punto en el gráfico denota un dato. Los colores indican la pertenencia al grupo determinado por K-Means.

Esta visualización nos brinda una idea de cómo los grupos se han formado en función de los resultados del algoritmo.

[34]:
```
# Visualizar los clusters con puntos más pequeños
plt.scatter(samsungData['Close'], samsungData['Volume'], c=samsungData['cluster'], cmap='rainbow', s=1)
plt.xlabel('Precio de Cierre')
plt.ylabel('Volumen')
plt.title('Agrupamiento de Datos Samsung')
plt.show()
```

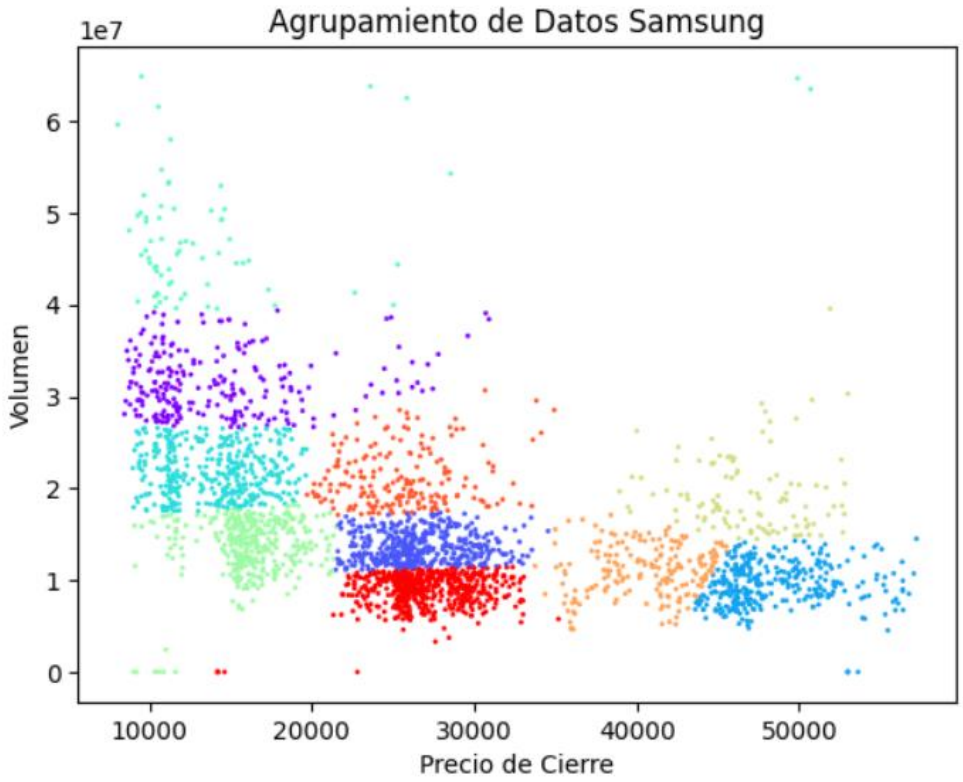


Enlace hacia un repositorio que contenga el modelo obtenido.

Para poder acceder a los datos y resultados, visitar el siguiente repositorio:

https://github.com/DiegoUlisesMM/Eva2BD

Gráfica personalizada e interpretación de resultados.



La gráfica de dispersión muestra la distribución de los datos de precios de cierre y volúmenes de acciones de Samsung en un espacio bidimensional. Cada punto en la gráfica representa una observación en el conjunto de datos. Los colores de los puntos indican a qué cluster ha sido asignada cada observación por el algoritmo K-Means.

En general, la agrupación proporciona una perspectiva interesante sobre cómo los precios de cierre y los volúmenes de negociación están relacionados en diferentes períodos de tiempo.

SA-4 Análisis no supervisado (comprar_alquilar)

4. Basado en el conjunto de datos "comprar_alquilar.csv" implemente el algoritmo de reducción de dimensionalidad de su preferencia y entregue un reporte que incluya:

Justificación del algoritmo.

El Análisis de Componentes Principales (PCA) es un algoritmo de reducción de dimensionalidad que se utiliza para transformar un conjunto de datos en un nuevo espacio de características de menor dimensión, preservando la mayor cantidad posible de la varianza original. Esto es útil para simplificar los datos y eliminar la multicolinealidad entre características, lo que puede mejorar la eficiencia de los modelos de aprendizaje automático y facilitar la visualización de los datos.

Descripción del diseño del modelo.

En este diseño, cargamos el archivo CSV "comprar_alquilar.csv" que contiene información sobre características relacionadas con la decisión de comprar o alquilar una propiedad.

[6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

[7]:
```
# Cargar el archivo CSV
dataAlquiler = pd.read_csv('C:/Users/Diego/OneDrive/Escritorio/DatosEvaluacion/comprar_alquilar.csv')
```

Eliminamos la columna de la variable objetivo "comprar" para realizar la reducción de dimensionalidad solo en las características.

[8]:
```
# Mostrar las primeras filas de los datos para explorar su estructura
print(dataAlquiler.head())
```

```
   ingresos  gastos_comunes  pago_coche  gastos_otros  ahorros  vivienda  \
0      6000            1000           0           600    50000    400000
1      6745             944         123           429    43240    636897
2      6455            1033          98           795    57463    321779
3      7098            1278          15           254    54506    660933
4      6167             863         223           520    41512    348932

   estado_civil  hijos  trabajo  comprar
0             0      2        2        1
1             1      3        6        0
2             2      1        8        1
3             0      0        3        0
4             0      0        3        1
```

[9]:
```
# Separar las características de la variable objetivo
X_alquiler = dataAlquiler.drop('comprar', axis=1)
y_alquiler = dataAlquiler['comprar']
```

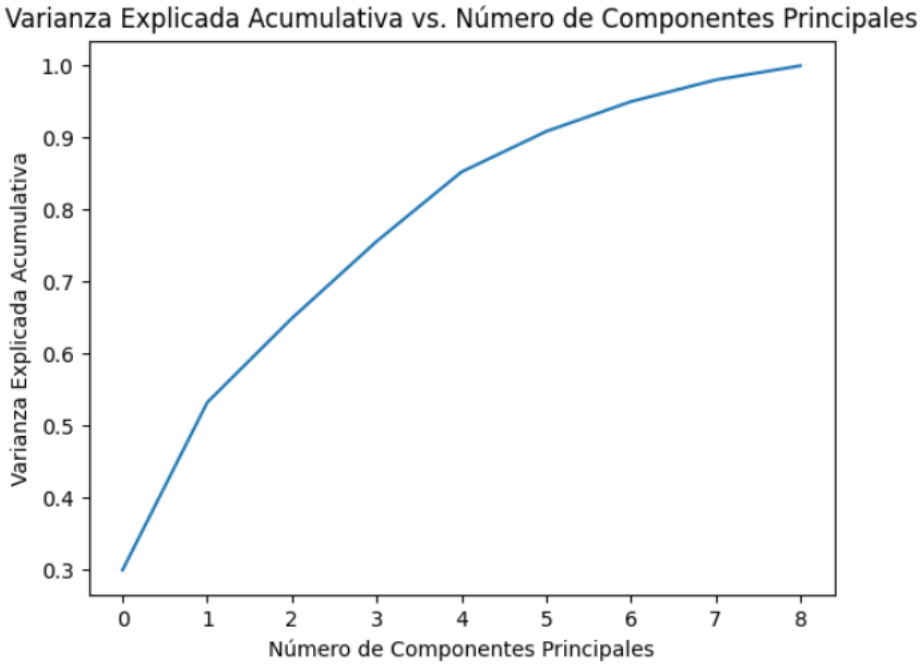
Normalizamos los datos para asegurarnos de que todas las características tengan el mismo peso en el proceso de reducción.

[10]:
```
# Normalizar los datos
scaler = StandardScaler()
X_scaled_alquiler = scaler.fit_transform(X_alquiler)
```

Luego, utilizamos PCA para transformar los datos normalizados en un nuevo espacio de características. Esto nos dará nuevas variables llamadas "componentes principales" que son combinaciones lineales de las características originales. Estos componentes están ordenados de manera que el primero captura la mayor varianza en los datos, el segundo el siguiente mayor y así sucesivamente.

[11]:
```
# Crear una instancia de PCA y ajustarla a los datos normalizados
pca = PCA()
X_pca_alquiler = pca.fit_transform(X_scaled_alquiler)
```

[12]:
```
# Graficar la varianza explicada acumulativa
explained_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
plt.plot(explained_variance_ratio)
plt.xlabel('Número de Componentes Principales')
plt.ylabel('Varianza Explicada Acumulativa')
plt.title('Varianza Explicada Acumulativa vs. Número de Componentes Principales')
plt.show()
```



Reducción de dimensionalidad

Se encuentra el número mínimo de componentes principales necesarios para retener un porcentaje específico de varianza explicada acumulativa. Este número se calcula utilizando np.argmax(explained_variance_ratio >= min_variance_ratio) + 1.

[17]:
```
# Encontrar el número mínimo de componentes para retener ese porcentaje de varianza
num_components = np.argmax(explained_variance_ratio >= min_variance_ratio) + 1
```

Se crea una nueva instancia de PCA con el número mínimo de componentes calculado en el paso anterior y se ajusta a los datos normalizados X_scaled_alquiler. Esto reduce las dimensiones de los datos al número mínimo de componentes y se muestra en consola

[12]:
```
# Aplicar la transformación PCA con el número mínimo de componentes
pca = PCA(n_components=num_components)
X_reduced = pca.fit_transform(X_scaled_alquiler)
```

[13]:
```
# Mostrar el número de componentes seleccionados
print(f"Número de componentes principales seleccionados: {num_components}")
```

```
Número de componentes principales seleccionados: 7
```

Se muestra la varianza explicada por cada componente principal utilizando un bucle.

[14]:
```
# Mostrar la varianza explicada por cada componente principal
print("Varianza explicada por cada componente principal:")
for i, variance_ratio in enumerate(pca.explained_variance_ratio_):
    print(f"Componente {i + 1}: {variance_ratio:.4f}")
```

```
Varianza explicada por cada componente principal:
Componente 1: 0.2991
Componente 2: 0.2329
Componente 3: 0.1171
Componente 4: 0.1069
Componente 5: 0.0964
Componente 6: 0.0563
Componente 7: 0.0415
```

Se calcula y muestra la varianza explicada acumulativa por número de componentes principales utilizando otro bucle.

[15]:
```
# Mostrar la varianza explicada acumulativa
explained_variance_cumulative = np.cumsum(pca.explained_variance_ratio_)
print("Varianza explicada acumulativa por número de componentes:")
for i, variance_cumulative in enumerate(explained_variance_cumulative):
    print(f"{i + 1} componentes: {variance_cumulative:.4f}")
```

```
Varianza explicada acumulativa por número de componentes:
1 componentes: 0.2991
2 componentes: 0.5321
3 componentes: 0.6492
4 componentes: 0.7561
5 componentes: 0.8524
6 componentes: 0.9087
7 componentes: 0.9502
```

Se elimina las dos columnas adicionales de X_pca_alquiler (PC8 y PC9) para que coincida con las 7 componentes principales seleccionadas y se crea nuevamente el DataFrame X_pca_alquiler con las columnas correctas para las 7 componentes principales.

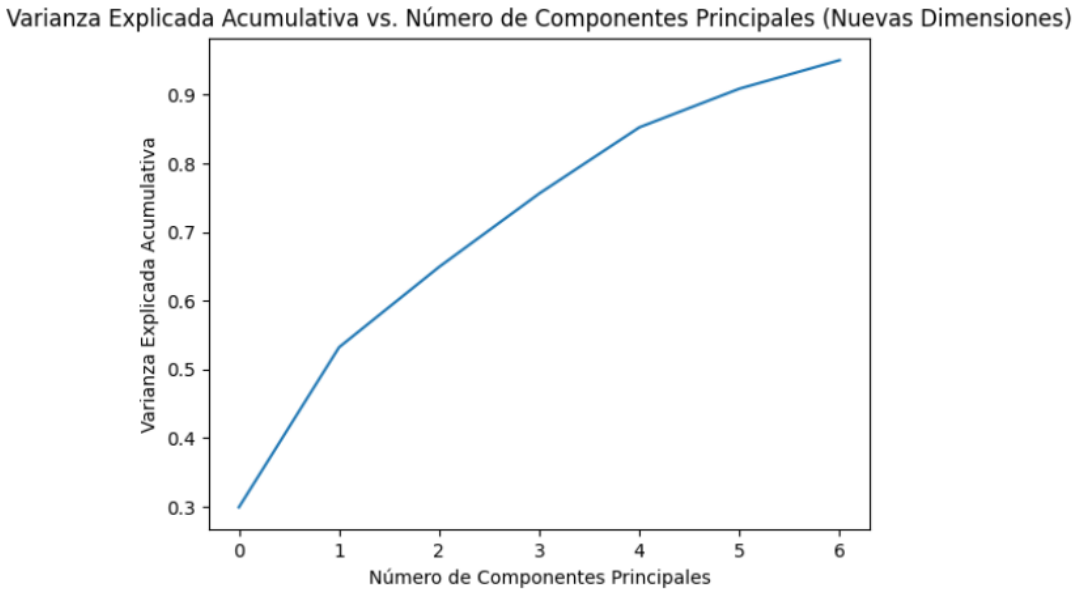
[33]:
```
# Eliminar las dos columnas adicionales de X_pca_alquiler
# mostramos los datos del entrenamiento
X_pca_alquiler = X_pca_alquiler[:, :7]

# Crear el DataFrame X_pca_alquiler con las columnas correctas
X_pca_alquiler = pd.DataFrame(
    X_pca_alquiler,
    columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'],
    index=dataAlquiler.index
)

X_pca_alquiler.head()
```

[33]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
0	-1.321915	-0.222950	-1.599863	-0.885014	0.646617	-0.700208	-0.318031
1	-1.147878	1.716967	-0.395722	-0.447960	-0.687805	-0.231969	-1.542487
2	-0.090369	1.836066	-1.359382	0.337505	0.742387	0.593030	1.540495
3	-3.455898	-0.078279	0.202382	-1.573584	-0.068551	-0.051945	-0.909409
4	-0.916933	-1.184676	-0.190036	0.117804	-0.147753	-0.619750	0.450419

Se grafica la varianza explicada acumulativa utilizando las 7 componentes principales en función del número de componentes.

[34]:
```
#Graficamos la varianza en sus nuevas dimensiones
explained_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
plt.plot(explained_variance_ratio)
plt.xlabel('Número de Componentes Principales')
plt.ylabel('Varianza Explicada Acumulativa')
plt.title('Varianza Explicada Acumulativa vs. Número de Componentes Principales (Nuevas Dimensiones)')
plt.show()
```

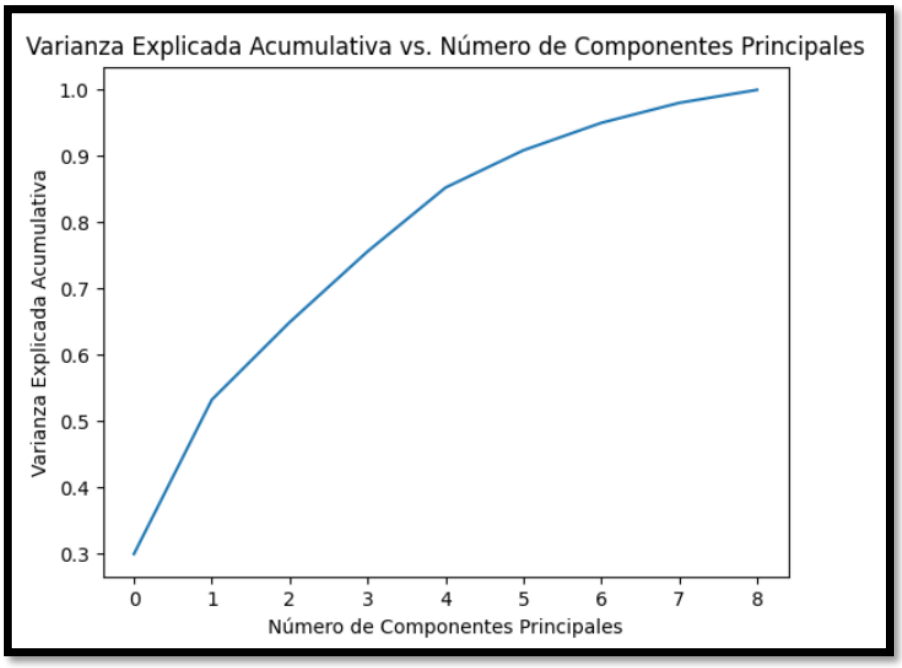


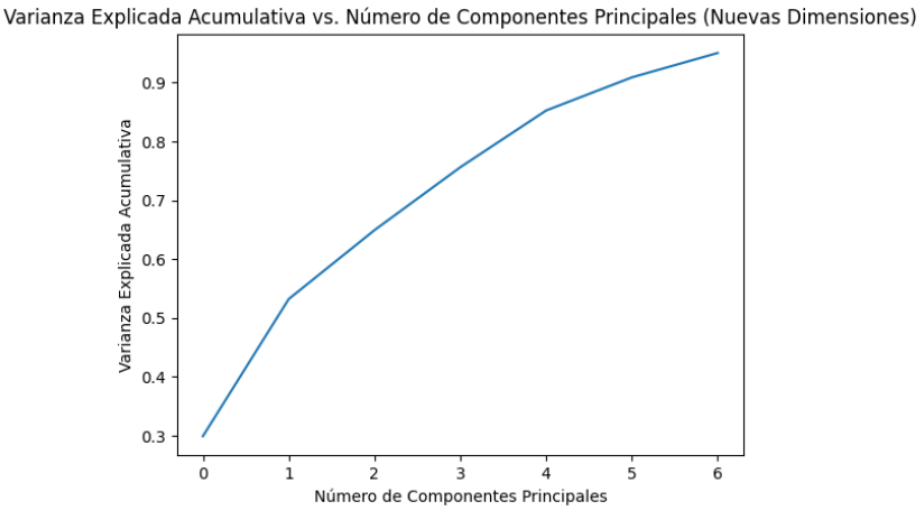
Enlace hacia un repositorio que contenga el modelo obtenido.

Para poder acceder a los datos y resultados, visitar el siguiente repositorio:

https://github.com/DiegoUlisesMM/Eva2BD

Gráfica personalizada e interpretación de resultados.





En la gráfica, el eje x representa el número de componentes principales utilizados, mientras que el eje y muestra la varianza explicada acumulativa hasta ese punto. La varianza explicada acumulativa indica cuánta información de los datos originales se mantiene al proyectarlos en un espacio de menor dimensión.

La siguiente grafica muestra cómo la información se distribuye en las nuevas dimensiones reducidas (componentes principales) después de eliminar las componentes menos significativas y retener solo aquellas que explican la mayor parte de la varianza en los datos. Comparando ambas gráficas, es posible observar cómo reducir la dimensionalidad a través de PCA puede afectar la cantidad de varianza explicada acumulativa y cómo las componentes principales seleccionadas influyen en la representación de los datos en menos dimensiones.

DE

Basado en el conjunto de datos "diabetes_indiana.csv" implemente un algoritmo de clasificación alterno al realizado en SA y entregue un reporte que incluya:

Justificación del algoritmo.

Gradient Boosting para este conjunto de datos debido a su capacidad para manejar relaciones no lineales entre características y etiquetas. Además, Gradient Boosting es robusto y puede trabajar bien incluso con datos ruidosos o con valores atípicos.

Descripción del diseño del modelo.

En este caso, utilizaremos la implementación de Gradient Boosting de la biblioteca scikit-learn para crear el modelo. Dividiremos el conjunto de datos en características (X) y etiquetas (y), y luego dividiremos estos datos en conjuntos de entrenamiento y prueba. A continuación, crearemos un modelo GradientBoostingClassifier, lo entrenaremos en el conjunto de entrenamiento y evaluaremos su rendimiento en el conjunto de prueba.

[29]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
```

[30]:
```
# Cargar los datos
data = pd.read_csv(r'C:\Users\Diego\OneDrive\Escritorio\DatosEvaluacion\diabetes_indiana.csv')
```

[31]:
```
# Dividir los datos en características (X) y etiquetas (y)
X = data.iloc[:, 1:-1]  # Excluir la columna "Unnamed: 0" y la columna de etiquetas
y = data.iloc[:, -1]    # Última columna (etiquetas)
```

[32]:
```
# Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[33]:
```
# Crear y entrenar el modelo Gradient Boosting
model = GradientBoostingClassifier()
model.fit(X_train, y_train)
```

[33]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

[34]:
```
# Predecir en el conjunto de prueba
y_pred = model.predict(X_test)
```

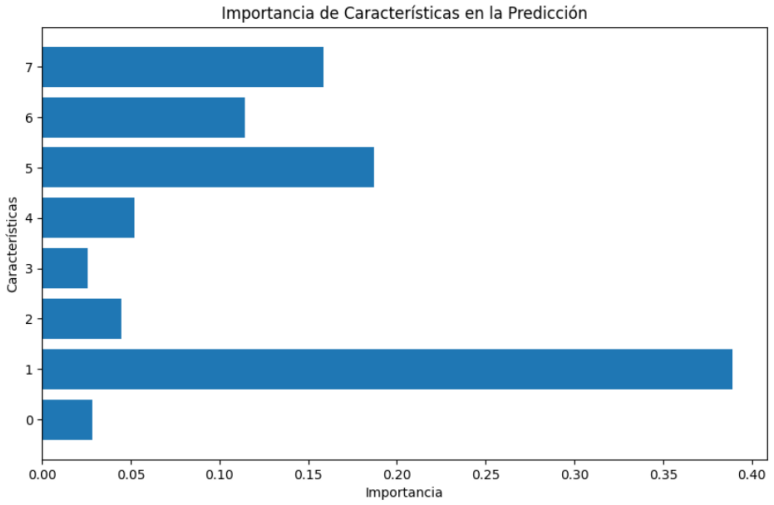
[35]:
```
# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

Evaluaremos el modelo utilizando métricas de evaluación, como precisión, recall y F1-score. También se ajustarán los hiperparámetros del modelo para mejorar su rendimiento

[36]:
```
# Imprimir métricas de evaluación
print(f'Precisión: {accuracy:.2f}')
print('Reporte de Clasificación:')
print(report)
```

```
Precisión: 0.74
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.81      0.78      0.79        99
           1       0.63      0.67      0.65        55

    accuracy                           0.74       154
   macro avg       0.72      0.73      0.72       154
weighted avg       0.75      0.74      0.74       154
```

[37]:
```
# Gráfica de importancia de características
feature_importances = model.feature_importances_
feature_names = X.columns
plt.figure(figsize=(10, 6))
plt.barh(feature_names, feature_importances)
plt.xlabel('Importancia')
plt.ylabel('Características')
plt.title('Importancia de Características en la Predicción')
plt.show()
```



Evaluación y optimización del modelo.

Ajustar los hiperparámetros del modelo puede mejorar su rendimiento. Uno de los enfoques comunes es realizar una búsqueda de cuadrícula (grid search) sobre diferentes combinaciones de hiperparámetros para encontrar la combinación óptima. A continuación, te proporciono los comandos para realizar una búsqueda de cuadrícula en el modelo Gradient Boosting

[38]:
```
from sklearn.model_selection import GridSearchCV
```

[39]:
```
# Definir los hiperparámetros a buscar
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
}
```

[40]:
```
# Crear el objeto GridSearchCV
grid_search = GridSearchCV(GradientBoostingClassifier(), param_grid, cv=5)
```

[41]:
```
# Realizar la búsqueda de cuadrícula en los datos de entrenamiento
grid_search.fit(X_train, y_train)
```

[41]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'learning_rate': [0.01, 0.1, 0.2],
                         'max_depth': [3, 4, 5],
                         'n_estimators': [50, 100, 200]})
```
GridSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier()
GradientBoostingClassifier
GradientBoostingClassifier()

[42]:
```
# Imprimir los resultados de la búsqueda de cuadrícula
print("Mejores hiperparámetros encontrados:")
print(grid_search.best_params_)
print("Mejor puntuación de validación cruzada:")
print(grid_search.best_score_)
```

```
Mejores hiperparámetros encontrados:
{'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 50}
Mejor puntuación de validación cruzada:
0.7817273090763694
```

[43]:
```
best_model = grid_search.best_estimator_
```

[44]:
```
# Predecir en el conjunto de prueba con el mejor modelo
y_pred_best = best_model.predict(X_test)
```

[45]:
```
# Evaluar el modelo ajustado
accuracy_best = accuracy_score(y_test, y_pred_best)
report_best = classification_report(y_test, y_pred_best)
```

[46]:
```
# Imprimir métricas de evaluación para el mejor modelo
print(f'Precisión del mejor modelo: {accuracy_best:.2f}')
print('Reporte de Clasificación para el mejor modelo:')
print(report_best)
```

```
Precisión del mejor modelo: 0.75
Reporte de Clasificación para el mejor modelo:
              precision    recall  f1-score   support

           0       0.82      0.79      0.80        99
           1       0.64      0.69      0.67        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.74       154
weighted avg       0.76      0.75      0.76       154
```

n_estimators: Este hiperparámetro se refiere al número de árboles de decisión que se van a construir en el proceso de boosting. Un valor más alto de n_estimators generalmente permite que el modelo sea más complejo y tenga un mayor poder predictivo. Sin embargo, un valor demasiado alto puede llevar a un sobreajuste. La búsqueda en la cuadrícula nos ayuda a encontrar un equilibrio adecuado para este valor, lo que puede conducir a un mejor rendimiento.

learning_rate: El learning rate (tasa de aprendizaje) controla la contribución de cada árbol al modelo general. Valores más bajos de learning rate hacen que el modelo aprenda más lentamente y puede requerir más árboles para ajustarse bien a los datos. Valores más altos pueden llevar a un ajuste excesivo. La búsqueda de cuadrícula nos permite encontrar el learning rate que resulta en un buen equilibrio entre convergencia rápida y evitando el sobreajuste.

max_depth: Este hiperparámetro controla la profundidad máxima de los árboles de decisión en el proceso de boosting. Árboles más profundos pueden capturar relaciones más complejas en los datos, pero también pueden llevar a un sobreajuste. Limitar la profundidad puede ayudar a prevenir el sobreajuste. La búsqueda en la cuadrícula nos ayuda a determinar la profundidad óptima que equilibra el poder predictivo con la generalización.

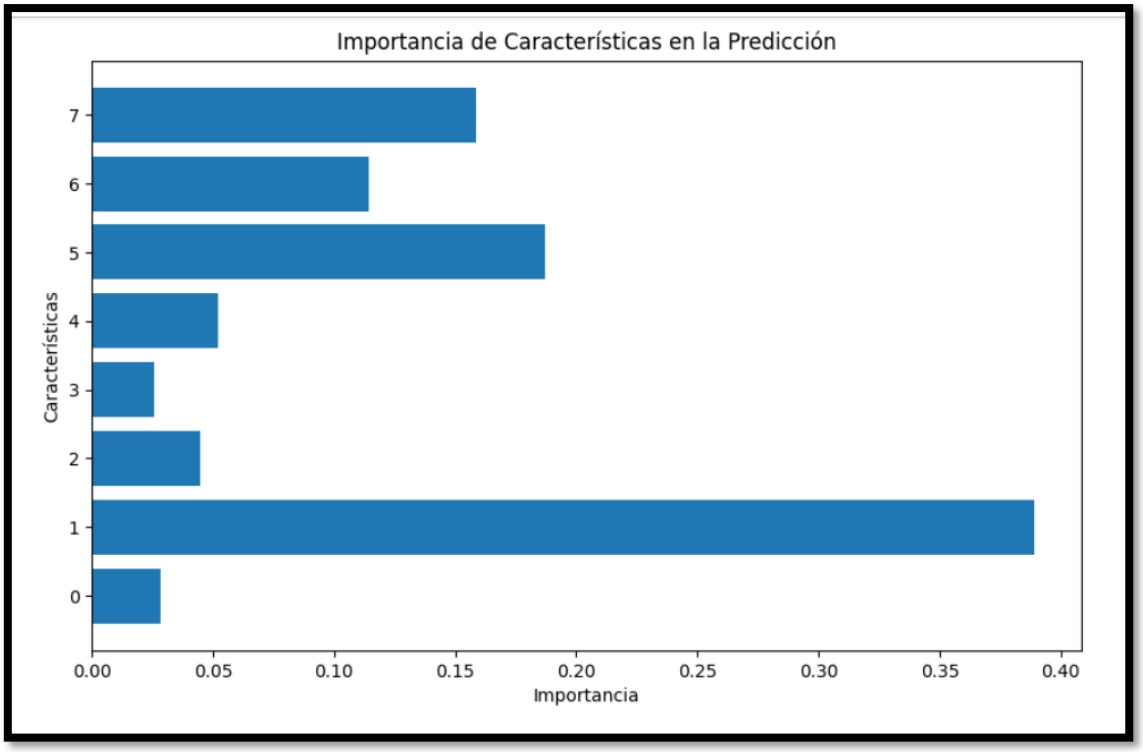
En resumen, al ajustar estos hiperparámetros, estamos controlando la complejidad y la velocidad de aprendizaje del modelo Gradient Boosting. La búsqueda de cuadrícula nos permite explorar diferentes combinaciones de estos hiperparámetros y encontrar el conjunto que optimiza el rendimiento del modelo en términos de precisión y capacidad de generalización en el conjunto de prueba.

Enlace hacia un repositorio que contenga el modelo obtenido.

Para poder acceder a los datos y resultados, visitar el siguiente repositorio:

https://github.com/DiegoUlisesMM/Eva2BD

Gráfica personalizada e interpretación de resultados.



La gráfica de importancia de características muestra la contribución relativa de cada característica en el proceso de toma de decisiones del modelo Gradient Boosting para predecir si una persona tiene diabetes o no. Las características con barras más largas son las que tuvieron una mayor influencia en la clasificación.

En el contexto de este conjunto de datos, la interpretación sería la siguiente:

Glucose (Glucosa): La concentración de glucosa en sangre es la característica más importante para determinar si una persona tiene diabetes. Valores más altos de glucosa tienden a aumentar la probabilidad de diabetes.

BMI (Índice de Masa Corporal): El índice de masa corporal también es un predictor significativo. Valores altos de BMI están correlacionados positivamente con la diabetes.

Age (Edad): La edad de la persona también tiene un impacto en la clasificación. Es posible que las personas mayores tengan una mayor probabilidad de desarrollar diabetes.

BloodPressure (Presión Arterial): Aunque menos influyente que las características anteriores, la presión arterial todavía contribuye en cierta medida a la clasificación.

SkinThickness (Espesor del Pliegue Cutáneo): Esta característica también influye en la clasificación, pero en menor medida en comparación con las anteriores.

El resto de las características parecen tener una influencia más baja en la clasificación según este modelo en particular.

AU

1. Elabora un documento donde implemente el aprendizaje por refuerzo (combinación de análisis supervisado y no supervisado)

Ejemplo básico que combina análisis no supervisado (K-Means) para la segmentación inicial de datos y análisis supervisado (SVM) para entrenar modelos en subconjuntos de datos segmentados. Luego, se combinan las decisiones de los modelos SVM entrenados en ambos clusters para obtener una estrategia de aprendizaje por refuerzo simple.

1. Carga de Datos: Se cargan los datos de diabetes_indiana.csv y se divide en características (X) y variable objetivo (y).

[48]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Cargar los datos
data_path = r'C:\Users\Diego\OneDrive\Escritorio\DatosEvaluacion\diabetes_indiana.csv'
diabetes_data = pd.read_csv(data_path)
```

2. Segmentación no Supervisada: Se aplica K-Means para dividir los datos en dos clusters, permitiendo la identificación de patrones no evidentes.

[49]:
```
# Dividir los datos en características (X) y variable objetivo (y)
X = diabetes_data.drop('8', axis=1)
y = diabetes_data['8']
```

3. Entrenamiento del Modelo SVM en Cluster 0: Se entrena un modelo SVM con kernel lineal en el subconjunto de datos pertenecientes al cluster 0.

[50]:
```
# Paso 1: Aplicar K-Means para segmentación no supervisada
kmeans = KMeans(n_clusters=2, random_state=42)
X['cluster'] = kmeans.fit_predict(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4. Evaluación del Modelo SVM en Cluster 0: Se evalúa el modelo SVM en el conjunto de prueba perteneciente al cluster 0 y se calcula su precisión.

[51]:
```
# Paso 2: Entrenar un modelo SVM en el subconjunto de cluster 0
X_cluster_0 = X_train[X_train['cluster'] == 0].drop('cluster', axis=1)
y_cluster_0 = y_train[X_train['cluster'] == 0]
svm_model_cluster_0 = SVC(kernel='linear')
svm_model_cluster_0.fit(X_cluster_0, y_cluster_0)
```

[51]:
```
SVC
SVC(kernel='linear')
```

5. Entrenamiento del Modelo SVM en Cluster 1: Se entrena un modelo SVM con kernel RBF en el subconjunto de datos pertenecientes al cluster 1.

[52]:
```
# Paso 3: Evaluar el modelo SVM en el conjunto de prueba
X_test_cluster_0 = X_test[X_test['cluster'] == 0].drop('cluster', axis=1)
y_test_cluster_0 = y_test[X_test['cluster'] == 0]
y_pred_cluster_0 = svm_model_cluster_0.predict(X_test_cluster_0)
accuracy_cluster_0 = accuracy_score(y_test_cluster_0, y_pred_cluster_0)

print("Precisión del Modelo SVM en Cluster 0:", accuracy_cluster_0)
```

```
Precisión del Modelo SVM en Cluster 0: 0.8
```

6. Evaluación del Modelo SVM en Cluster 1: Se evalúa el modelo SVM en el conjunto de prueba perteneciente al cluster 1 y se calcula su precisión.

[53]:
```
# Paso 4: Entrenar un modelo SVM en el subconjunto de cluster 1
X_cluster_1 = X_train[X_train['cluster'] == 1].drop('cluster', axis=1)
y_cluster_1 = y_train[X_train['cluster'] == 1]
svm_model_cluster_1 = SVC(kernel='rbf')
svm_model_cluster_1.fit(X_cluster_1, y_cluster_1)
```

[53]:
```
▾ SVC
SVC()
```

7. Combinación de Predicciones: Se combinan las predicciones de ambos modelos SVM basados en los clusters para obtener un modelo final.

[54]:
```
# Paso 5: Evaluar el modelo SVM en el conjunto de prueba
X_test_cluster_1 = X_test[X_test['cluster'] == 1].drop('cluster', axis=1)
y_test_cluster_1 = y_test[X_test['cluster'] == 1]
y_pred_cluster_1 = svm_model_cluster_1.predict(X_test_cluster_1)
accuracy_cluster_1 = accuracy_score(y_test_cluster_1, y_pred_cluster_1)

print("Precisión del Modelo SVM en Cluster 1:", accuracy_cluster_1)
```

```
Precisión del Modelo SVM en Cluster 1: 0.6962025316455697
```

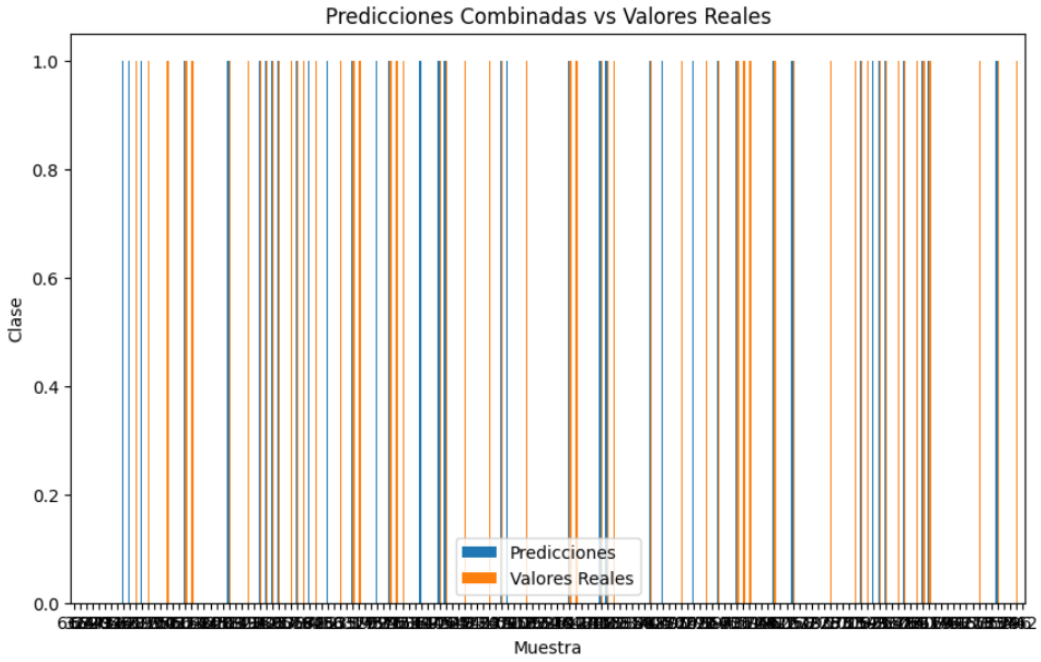
8. Evaluación del Modelo Combinado: Se evalúa el modelo final combinado en todo el conjunto de prueba y se calcula su precisión.

[64]:
```
# Paso 6: Combinar las decisiones del modelo SVM en ambos clusters
combined_predictions = pd.Series(index=X_test.index)
combined_predictions[X_test_cluster_0.index] = y_pred_cluster_0
combined_predictions[X_test_cluster_1.index] = y_pred_cluster_1

# Paso 7: Evaluar el modelo final combinado en todo el conjunto de prueba
accuracy_combined = accuracy_score(y_test, combined_predictions)
print("Precisión del Modelo Combinado:", accuracy_combined)
```

```
Precisión del Modelo Combinado: 0.7467532467532467
```

9. Visualización de Resultados: Se crea un gráfico de barras que muestra las predicciones combinadas y los valores reales en la muestra.

[72]:
```
import matplotlib.pyplot as plt

# Crear un DataFrame con las predicciones combinadas y los valores reales
results = pd.DataFrame({'Predicciones': combined_predictions, 'Valores Reales': y_test})

# Graficar las predicciones combinadas y los valores reales
results.plot(kind='bar', figsize=(10, 6))
plt.title('Predicciones Combinadas vs Valores Reales')
plt.xlabel('Muestra')
plt.ylabel('Clase')
plt.xticks(rotation=0)
plt.legend()
plt.show()
```



El gráfico final muestra la comparación entre las predicciones combinadas y los valores reales para cada muestra en el conjunto de prueba. Cada barra representa la predicción combinada y el valor real para una muestra específica. El análisis de este gráfico permite observar cómo se desempeñó el modelo combinado en comparación con los valores reales en cada caso.

Este proceso demuestra cómo se puede aprovechar tanto el análisis no supervisado (K-Means) como el análisis supervisado (SVM) para mejorar la capacidad de predicción en un conjunto de datos.