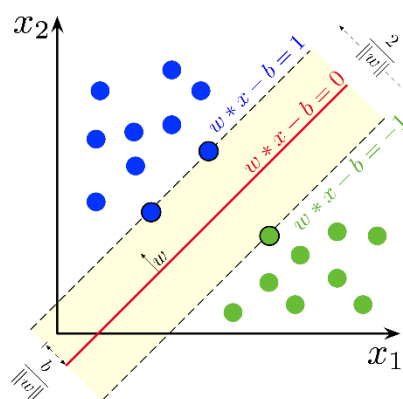




**Centro Universitario de Ciencias Exactas e
Ingenierías**
Universidad de Guadalajara



Práctica 6: SVM
Aprendizaje Máquina



Alumno: Samuel David Pérez Brambila

Código: 222966286

Profesora: Karla Ávila Cárdenas

Sección: D01

Fecha de Entrega: 20 de Octubre de 2024

Introducción

Support vector machine (SVM) es “un algoritmo de aprendizaje supervisado que se utiliza en muchos problemas de clasificación y regresión, incluidas aplicaciones médicas de procesamiento de señales, procesamiento del lenguaje natural y reconocimiento de imágenes y voz.” (MathWorks, s.f.)

De acuerdo con MathWorks (s.f.), el objetivo del algoritmo SVM es encontrar un hiperplano que separe de la mejor forma posible dos clases diferentes de puntos de datos. “De la mejor forma posible” implica el hiperplano con el margen más amplio entre las dos clases. El margen se define como la anchura máxima de la región paralela al hiperplano que no tiene puntos de datos interiores. El algoritmo solo puede encontrar este hiperplano en problemas que permiten separación lineal; en la mayoría de los problemas prácticos, el algoritmo maximiza el margen flexible permitiendo un pequeño número de clasificaciones erróneas.

Así mismo, MathWorks (s.f.) nos deja en claro que los vectores de soporte hacen referencia a un subconjunto de las observaciones de entrenamiento que identifican la ubicación del hiperplano de separación, además support vector machines pertenecen a una clase de algoritmos de Machine Learning denominados métodos kernel, donde se puede utilizar una función de kernel para transformar las características. Las funciones de kernel asignan los datos a un espacio dimensional diferente, que suele ser superior, con la expectativa de que resulte más fácil separar las clases después de esta transformación, simplificando potencialmente los límites de decisión complejos no lineales para hacerlos lineales en el espacio dimensional de características superior asignado. En este proceso, los datos no se tienen que transformar explícitamente, lo que supondría una alta carga computacional. Esto se conoce como truco de kernel.

Existen varios tipos de SVM, dependiendo del kernel que se utilice. Algunos de los más comunes son:

Tipo de SVM	Kernel	Descripción
Función de base radial (RBF) o gaussiana	$K(x_1, x_2) = \exp\left(-\frac{\ x_1 - x_2\ ^2}{2\sigma^2}\right)$	Aprendizaje de una clase. σ representa la anchura del kernel.
Lineal	$K(x_1, x_2) = x_1^T x_2$	Aprendizaje de dos clases.
Polinómica	$K(x_1, x_2) = (x_1^T x_2 + 1)^\rho$	ρ representa el orden del polinomio.
Sigmoide	$K(x_1, x_2) = \tanh(\beta_0 x_1^T x_2 + \beta_1)$	Representa un kernel de Mercer solo para determinados valores β_0 y β_1 .

Según InteractiveChaos (s.f.), este tipo de algoritmos destacan por:

- Ser efectivos en espacios de alta dimensionalidad, aun cuando el número de dimensiones supera el número de muestras.
- Eficiente gestión de la memoria, al usar solo un subconjunto de puntos en la función de decisión.

Las desventajas, por el contrario, incluyen las siguientes:

- Su eficacia depende del kernel que se escoja.
- Resultan poco eficientes con datasets grandes (el cálculo del kernel puede resultar muy lento).
- Si el número de características es mucho mayor que el número de muestras resulta crucial evitar el sobreentrenamiento escogiendo el kernel y el término de regularización adecuados.
- SVM no proporciona estimaciones de probabilidad.
- La frontera de decisión depende directamente de los valores más próximos, aunque sean erróneos.
- SVM es muy dependiente de la escala de los datos, por lo que convendrá escalarlos adecuadamente.

En la siguiente práctica, se implementará un SVM lineal utilizando el dataset de cáncer de mama proporcionado por la librería sklearn en Python. Para desarrollar este modelo, emplearemos librerías adicionales como Numpy, Seaborn, Pandas, Matplotlib, entre otras, que facilitarán la carga, procesamiento y visualización de los datos.

Contenido de la Actividad

El presente código implementa el algoritmo de Máquinas de Vectores de Soporte (SVM) sobre el conjunto de datos "Breast Cancer Wisconsin", que se utiliza para clasificar tumores como benignos o malignos. Se emplean las características "mean radius" (radio medio del tumor), "mean texture" (textura media del tumor), y se evalúa el rendimiento del modelo utilizando un enfoque de validación cruzada. El modelo SVM ajusta un hiperplano de separación entre las clases, permitiendo clasificar nuevos datos basándose en la distancia a dicho hiperplano. Además, se optimiza el modelo explorando diferentes valores del parámetro de regularización C para encontrar el más adecuado, logrando un balance entre la complejidad del modelo y su capacidad de generalización. La evaluación se realiza mediante métricas como la precisión, el recall y la matriz de confusión, lo que proporciona una visión clara del desempeño del modelo en la clasificación de los tumores.

Importación de bibliotecas para tratamiento de datos

```
Prácticas - practica6_SVM.py

1 # Tratamiento de datos
2 # =====
3 import pandas as pd
4 import numpy as np
5 from sklearn.datasets import load_breast_cancer
```

- import pandas as pd: Se importa la biblioteca pandas para manejar estructuras de datos y realizar análisis de datos.
- import numpy as np: Se importa numpy, que es útil para realizar cálculos numéricos y operaciones con matrices.
- from sklearn.datasets import load_breast_cancer: Importa una función específica de scikit-learn que permite cargar el conjunto de datos de cáncer de mama.

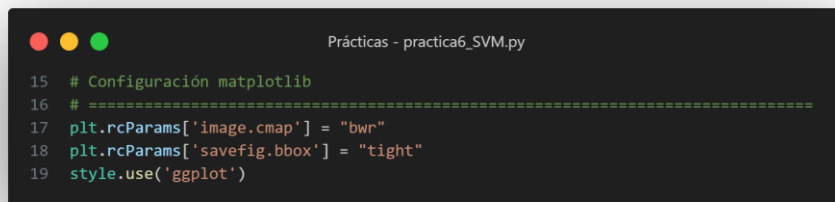
Importación de bibliotecas para gráficos y métricas

```
Prácticas - practica6_SVM.py

7 # Gráficos y métricas
8 # =====
9 import matplotlib.pyplot as plt
10 from matplotlib import style
11 from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
12 from sklearn.svm import SVC
13 from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, cross_val_predict
```

- import matplotlib.pyplot as plt: Importa matplotlib, que es una biblioteca para crear gráficos.
- from matplotlib import style: Permite aplicar estilos predefinidos a los gráficos.
- from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score: Importa métricas de evaluación, como la matriz de confusión, precisión, exactitud y recall.
- from sklearn.svm import SVC: Importa el clasificador SVM (Máquinas de Vectores de Soporte).
- from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, cross_val_predict: Importa herramientas para dividir el conjunto de datos y realizar búsqueda de hiperparámetros.

Configuración de Matplotlib:



```

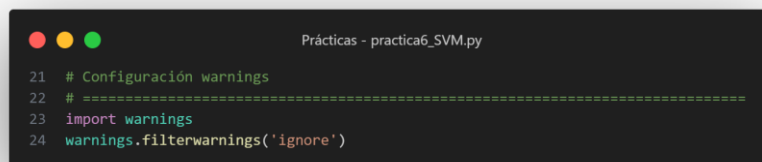
Prácticas - practica6_SVM.py

15 # Configuración matplotlib
16 # =====
17 plt.rcParams['image.cmap'] = "bwr"
18 plt.rcParams['savefig.bbox'] = "tight"
19 style.use('ggplot')

```

- plt.rcParams['image.cmap'] = "bwr": Establece el mapa de colores para los gráficos.
- plt.rcParams['savefig.bbox'] = "tight": Asegura que las figuras se guarden sin espacios adicionales alrededor.
- style.use('ggplot'): Aplica el estilo de ggplot a los gráficos, que es visualmente atractivo y fácil de interpretar.

Configuración de warnings:



```

Prácticas - practica6_SVM.py

21 # Configuración warnings
22 # =====
23 import warnings
24 warnings.filterwarnings('ignore')

```

- import warnings: Importa la biblioteca para gestionar advertencias.
- warnings.filterwarnings('ignore'): Suprime las advertencias para que no se muestren en la salida.

Carga del dataset

```
Prácticas - practica6_SVM.py
26 # Carga del Dataset Breast Cancer Wisconsin
27 # =====
28 data = load_breast_cancer()
29 X = pd.DataFrame(data['data'], columns=data['feature_names'])
30 y = data['target']
```

- data = load_breast_cancer(): Carga el conjunto de datos de cáncer de mama en la variable data.
- X = pd.DataFrame(data['data'], columns=data['feature_names']): Crea un DataFrame de pandas para las características (X) utilizando los nombres de las características.
- y = data['target']: Extrae las etiquetas de clase (benigno o maligno) en y.

Selección de características

```
Prácticas - practica6_SVM.py
32 # Usaremos dos características para graficar
33 X = X[['mean radius', 'mean texture']]
```

- X = X[['mean radius', 'mean texture']]: Reduce X a solo dos características (mean radius y mean textura) para facilitar la visualización en gráficos.

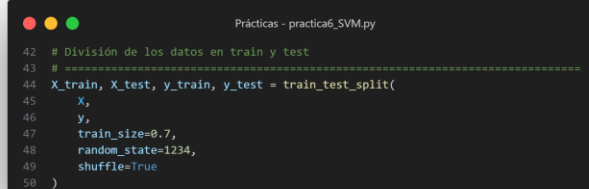
Visualización de Datos

```
Prácticas - practica6_SVM.py
35 # Visualización de los datos
36 # =====
37 fig, ax = plt.subplots(figsize=(6, 4))
38 ax.scatter(X['mean radius'], X['mean texture'], c=y)
39 ax.set_title("Breast Cancer Dataset (Radio Medio vs Textura Media)")
40 plt.show()
```

- fig, ax = plt.subplots(figsize=(6, 4)): Crea una figura y un eje para graficar con un tamaño específico.
- ax.scatter(X['mean radius'], X['mean texture'], c=y): Crea un gráfico de dispersión utilizando las características seleccionadas y colorea los puntos según las etiquetas y.

- ax.set_title('Breast Cancer Dataset (Radio medio vs Textura media): Establece el título del gráfico.
- plt.show(): Muestra el gráfico en pantalla.

División de Datos en Train y Test



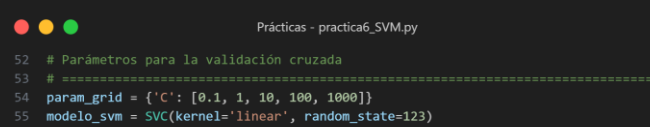
```

Prácticas - practica6_SVM.py
42 # División de los datos en train y test
43 # =====
44 X_train, X_test, y_train, y_test = train_test_split(
45     X,
46     y,
47     train_size=0.7,
48     random_state=1234,
49     shuffle=True
50 )

```

- X_train, X_test, y_train, y_test = train_test_split(...): Divide el conjunto de datos en entrenamiento (70%) y prueba (30%). random_state asegura que la división sea reproducible y shuffle=True mezcla los datos antes de la división.

Parámetros para Validación Cruzada



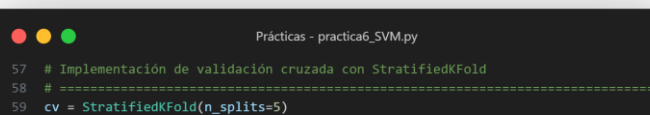
```

Prácticas - practica6_SVM.py
52 # Parámetros para la validación cruzada
53 # =====
54 param_grid = {'C': [0.1, 1, 10, 100, 1000]}
55 modelo_svm = SVC(kernel='linear', random_state=123)

```

- param_grid = {'C': ...}: Define un diccionario con los valores del parámetro C que se probarán durante la búsqueda de hiperparámetros. C controla la regularización en SVM.
- modelo_svm = SVC(kernel='linear', random_state=123): Crea una instancia del clasificador SVM con un kernel lineal.

Validación Cruzada con K-folds con StratifiedKFold



```

Prácticas - practica6_SVM.py
57 # Implementación de validación cruzada con StratifiedKFold
58 # =====
59 cv = StratifiedKFold(n_splits=5)

```

- cv = StratifiedKFold(n_splits=5): Crea un objeto de validación cruzada que divide los datos en 5 pliegues, asegurando que cada pliegue tenga una proporción similar de clases.

Búsqueda del mejor modelo

```
Prácticas - practica6_SVM.py
61 # GridSearchCV para buscar el mejor modelo
62 # =====
63 grid_search = GridSearchCV(estimator=modelo_svm, param_grid=param_grid, scoring='accuracy', cv=cv)
64 grid_search.fit(X_train, y_train)
```

- grid_search = GridSearchCV(...): Crea un objeto para realizar una búsqueda de hiperparámetros utilizando validación cruzada.
- grid_search.fit(X_train, y_train): Ajusta el modelo SVM a los datos de entrenamiento y encuentra el mejor parámetro C.

Evaluación por cada valor de C

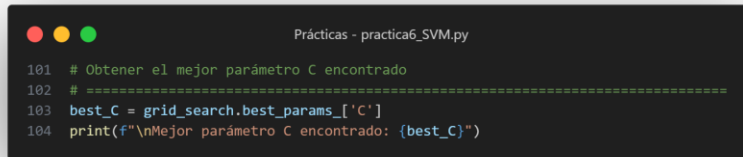
```
Prácticas - practica6_SVM.py
66 # Evaluación para cada valor de C en la validación cruzada con K-folds
67 # =====
68
69 import seaborn as sns # Importa la librería seaborn
70
71 for C_value in param_grid['C']:
72     modelo_temp = SVC(kernel='linear', C=C_value, random_state=123)
73     y_pred_cv = cross_val_predict(modelo_temp, X_train, y_train, cv=cv)
74
75     # Matriz de confusión
76     conf_matrix = confusion_matrix(y_train, y_pred_cv)
77
78     # Cálculo de accuracy, precisión y recall
79     accuracy = accuracy_score(y_train, y_pred_cv)
80     precision = precision_score(y_train, y_pred_cv, average='weighted')
81     recall = recall_score(y_train, y_pred_cv, average='weighted')
82
83     # Mostrar los resultados de cada iteración
84     print(f"\n--- Evaluación del Modelo SVM con C={C_value} ---")
85     print(f"Accuracy: {accuracy * 100:.2f}%")
86     print(f"Precisión: {precision * 100:.2f}%")
87     print(f"Recall: {recall * 100:.2f}%")
88     print("Matriz de Confusión:")
89     print(conf_matrix)
90
91     # Visualización gráfica de la matriz de confusión
92     plt.figure(figsize=(6, 5)) # Configura el tamaño de la figura
93     sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
94                 xticklabels=['Benigno', 'Maligno'],
95                 yticklabels=['Benigno', 'Maligno']) # Etiquetas para las clases
96     plt.title(f'Matriz de Confusión (C={C_value})')
97     plt.xlabel('Predicción')
98     plt.ylabel('Real')
99     plt.show() # Muestra la gráfica
```

- Importación de Seaborn: Se importa la biblioteca seaborn, que se usará más adelante para crear mapas de calor (heatmaps) para visualizar la matriz de confusión.

- Bucle For: Se inicia un bucle que iterará sobre cada valor de C que se ha definido previamente en `param_grid['C']`. Este bucle se usa para evaluar cómo cada valor de C afecta al rendimiento del modelo.
- Creación del Modelo SVM (modelo temp): Se crea una instancia temporal del modelo SVM utilizando el kernel lineal y el valor de C actual del bucle. `random_state=123` asegura que los resultados sean reproducibles.
- Predicción con Validación Cruzada (y_pred_cv): Se usa la función `cross_val_predict` para realizar la validación cruzada. Esto entrenará el modelo en diferentes pliegues (folds) y hará predicciones para cada muestra en `X_train`, devolviendo las predicciones en `y_pred_cv`.
- Cálculo de la Matriz de Confusión (conf_matrix): Se calcula la matriz de confusión usando las verdaderas etiquetas `y_train` y las predicciones `y_pred_cv`. La matriz de confusión muestra el rendimiento del modelo en términos de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos.
- Cálculo de Métricas de Rendimiento: Se calculan varias métricas para evaluar el rendimiento del modelo:
 - Accuracy: Porcentaje de predicciones correctas en comparación con el total de predicciones.
 - Precisión: Proporción de verdaderos positivos respecto a la suma de verdaderos positivos y falsos positivos, ponderada por la cantidad de muestras en cada clase.
 - Recall: Proporción de verdaderos positivos respecto a la suma de verdaderos positivos y falsos negativos, también ponderada por la cantidad de muestras en cada clase.
- Impresión de Resultados: Se imprimen los resultados de la evaluación del modelo para el valor actual de C :
 - Se imprime el valor de C en uso.
 - Se muestra la precisión, el recall y la matriz de confusión.
- Configuración de la Figura (plt.figure(figsize=(6,5))): Se crea una nueva figura con un tamaño específico (6x5 pulgadas) para la visualización de la matriz de confusión.
- Creación del Mapa de Calor: Se usa `sns.heatmap()` para crear un mapa de calor basado en la matriz de confusión:
 - annot=True: Muestra los valores numéricos dentro de cada celda del mapa.
 - fmt='d': Formato de los valores que se muestran, en este caso enteros.
 - cmap='Blues': Esquema de colores para el mapa de calor.
 - xticklabels y yticklabels: Se etiquetan los ejes X e Y con los nombres de las clases (Benigno y Maligno).
- Etiquetas y Título: Se añaden un título y etiquetas a los ejes:
 - El título incluye el valor de C para referenciar la matriz de confusión correspondiente.

- Se etiquetan los ejes X como "Predicción" y el eje Y como "Real" para clarificar el significado de cada eje.
- Mostrar Gráfica: Finalmente, se muestra la gráfica generada en la figura. Esto permite visualizar la matriz de confusión en un formato gráfico, lo que facilita la interpretación de los resultados.

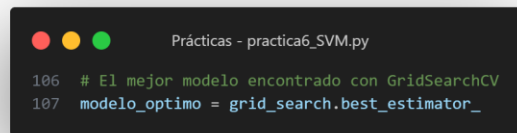
Mejor parámetro C encontrado



```
Prácticas - practica6_SVM.py
101 # Obtener el mejor parámetro C encontrado
102 # =====
103 best_C = grid_search.best_params_['C']
104 print(f"\nMejor parámetro C encontrado: {best_C}")
```

- best_C = grid_search.best_params_['C']: Obtiene el mejor valor de C encontrado por GridSearchCV.
- Impresión del mejor C: Muestra el mejor parámetro encontrado.

Modelo óptimo encontrado



```
Prácticas - practica6_SVM.py
106 # El mejor modelo encontrado con GridSearchCV
107 modelo_optimo = grid_search.best_estimator_
```

- modelo_optimo = grid_search.best_estimator_ : Guarda el mejor modelo ajustado que se encontró durante la búsqueda.

Evaluación Final en el Conjunto de Test

```
Prácticas - practica6_SVM.py

109 # Evaluación final en el conjunto de test
110 # =====
111 y_pred_test = modelo_optimo.predict(X_test)
112
113 # Matriz de confusión para el conjunto de test
114 conf_matrix_test = confusion_matrix(y_test, y_pred_test)
115 accuracy_test = accuracy_score(y_test, y_pred_test)
116 precision_test = precision_score(y_test, y_pred_test, average='weighted')
117 recall_test = recall_score(y_test, y_pred_test, average='weighted')
118
119 print("\n--- Evaluación final en el conjunto de test ---")
120 print(f"Accuracy: {accuracy_test * 100:.2f}%")
121 print(f"Precisión: {precision_test * 100:.2f}%")
122 print(f"Recall: {recall_test * 100:.2f}%")
123 print("Matriz de Confusión (test):")
124 print(conf_matrix_test)
125
126 plt.figure(figsize=(6, 5)) # Configura el tamaño de la figura
127 sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues',
128             xticklabels=['Benigno', 'Maligno'],
129             yticklabels=['Benigno', 'Maligno']) # Etiquetas para las clases
130 plt.title('Matriz de Confusión en el Conjunto de Test')
131 plt.xlabel('Predicción')
132 plt.ylabel('Real')
133 plt.show() # Muestra la gráfica
```

- y_pred_test = modelo_optimo.predict(X_test): Realiza predicciones sobre el conjunto de prueba.
- Cálculo de métricas: Calcula la matriz de confusión, exactitud, precisión y recall para evaluar el rendimiento del modelo en el conjunto de prueba.
- Impresión de resultados: Muestra la evaluación final del modelo.
- Al igual que el anterior, se crea un heatmap para mostrar la matriz de confusión de manera gráfica.

Representación Gráfica de Límites de Clasificación

```
Prácticas - practica6_SVM.py

135 # Representación gráfica de los límites de clasificación para el mejor modelo
136 # =====
137 # Grid de valores para graficar
138 x = np.linspace(np.min(X_train['mean radius']) - 7, np.max(X_train['mean radius']) + 7, 50)
139 y = np.linspace(np.min(X_train['mean texture']) - 7, np.max(X_train['mean texture']) + 7, 50)
140 X_grid = np.meshgrid(x, y)
141 grid = np.vstack([X_grid.ravel(), Y.ravel()]).T
```

- Creación de un grid: Se crean arrays x e y con valores que cubren el rango de las características en el conjunto de entrenamiento.
- np.meshgrid(...): Genera una cuadrícula de puntos (X, Y) a partir de x e y.
- grid = np.vstack([...]).T: Combina las coordenadas del grid en una sola matriz 2D.

Predicción de Clases para el Grid

```
Prácticas - practica6_SVM.py

143 # Predicción de las clases para los puntos del grid
144 pred_clases_grid = modelo_optimo.predict(grid)
145
146 # Reshape de las predicciones para que coincidan con la forma del grid
147 Z = pred_clases_grid.reshape(X_grid.shape)
```

- pred_clases_grid = modelo_optimo.predict(grid): Predice las clases para todos los puntos en el grid.
- Z = pred_clases_grid.reshape(X_grid.shape): Da forma a las predicciones para que coincidan con la cuadrícula de puntos.

Graficar resultados

```
Prácticas - practica6_SVM.py

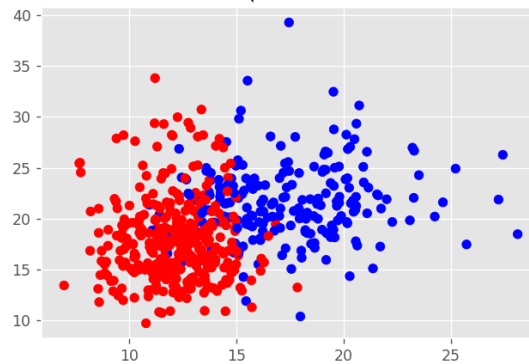
149 # Graficar resultados
150 fig, ax = plt.subplots(figsize=(8, 6))
151
152 # Usar contourf para rellenar las áreas de las clases (rojo y azul)
153 contour = ax.contourf(X_grid, Y, Z, alpha=0.3, cmap='coolwarm')
154
155 # Graficar las predicciones del grid con transparencia
156 ax.scatter(grid[:, 0], grid[:, 1], c=pred_clases_grid, alpha=0.2, cmap='coolwarm')
157
158 # Graficar los datos de entrenamiento con las clases reales
159 scatter = ax.scatter(X_train['mean radius'], X_train['mean texture'], c=y_train, edgecolor='k', cmap='coolwarm')
160
161 # Vectores soporte
162 ax.scatter(
163     modelo_optimo.support_vectors[:, 0],
164     modelo_optimo.support_vectors[:, 1],
165     s=200, linewidth=1,
166     facecolors='none', edgecolors='black'
167 )
168
169 # Hiperplano de separación
170 ax.contour(
171     X_grid, Y, modelo_optimo.decision_function(grid).reshape(X_grid.shape),
172     colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--']
173 )
174
175 # Agregar leyenda que explique los colores
176 from matplotlib.lines import Line2D
177 legend_elements = [
178     Line2D([0], [0], marker='o', color='w', label='Benigno', markerfacecolor='blue', markersize=10),
179     Line2D([0], [0], marker='o', color='w', label='Maligno', markerfacecolor='red', markersize=10),
180 ]
181
182 ax.legend(handles=legend_elements, loc='upper right')
183
184 ax.set_title("Resultados clasificación SVM lineal (Breast Cancer Dataset)")
185 ax.set_xlabel("Radio Medio")
186 ax.set_ylabel("Textura Media")
187 plt.show()
```

- Creación de un gráfico: Se inicializa la figura y los ejes para graficar.
- contourf(...): Dibuja contornos rellenos para mostrar las áreas de clasificación del modelo.
- ax.scatter(...): Grafica los puntos del grid con una baja opacidad.
- scatter = ax.scatter(...): Grafica los datos de entrenamiento con los colores que representan sus clases reales.

- Vectores de soporte: Grafica los vectores de soporte del modelo, resaltándolos con un borde negro.
- Hiperplano de separación: Grafica las líneas que representan el hiperplano de separación.
- Leyenda: Crea una leyenda para explicar los colores de las clases.
- Etiquetas y título: Establece las etiquetas para los ejes y el título del gráfico.
- plt.show(): Muestra el gráfico final.

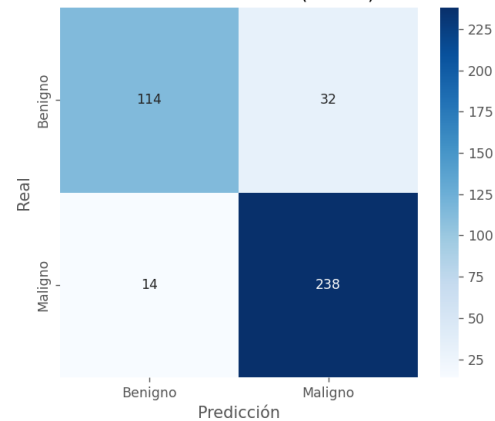
Es así, que obtenemos los siguientes resultados:

Breast Cancer Dataset (Radio Medio vs Textura Media)



```
--- Evaluación del Modelo SVM con C=0.1 ---
Accuracy: 88.44%
Precisión: 88.48%
Recall: 88.44%
Matriz de Confusión:
[[114  32]
 [ 14 238]]
```

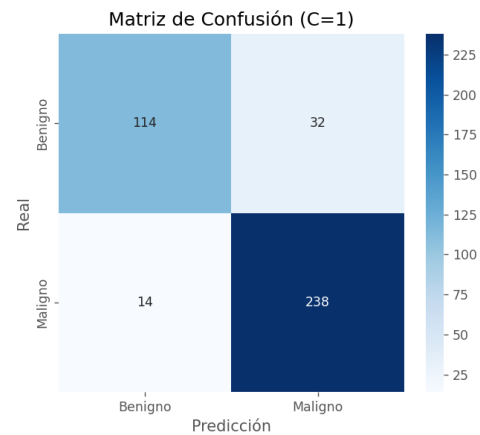
Matriz de Confusión (C=0.1)



```

--- Evaluación del Modelo SVM con C=1 ---
Accuracy: 88.44%
Precisión: 88.48%
Recall: 88.44%
Matriz de Confusión:
[[114  32]
 [ 14 238]]

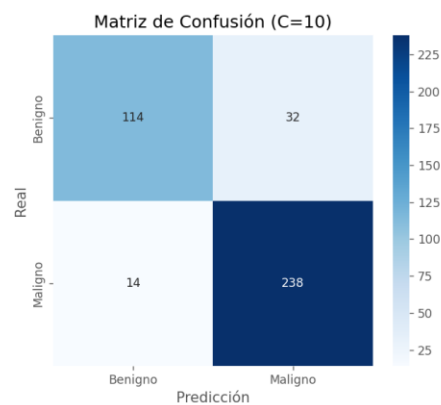
```



```

--- Evaluación del Modelo SVM con C=10 ---
Accuracy: 88.44%
Precisión: 88.48%
Recall: 88.44%
Matriz de Confusión:
[[114  32]
 [ 14 238]]

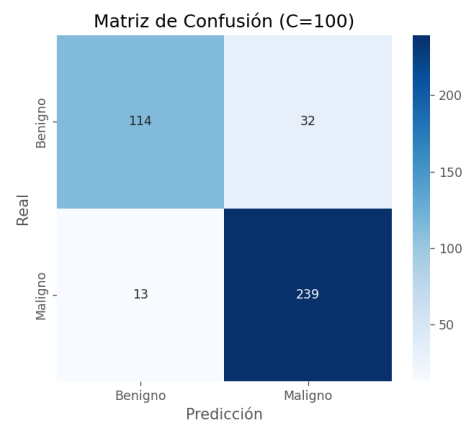
```



```

--- Evaluación del Modelo SVM con C=100 ---
Accuracy: 88.69%
Precisión: 88.77%
Recall: 88.69%
Matriz de Confusión:
[[114  32]
 [ 13 239]]

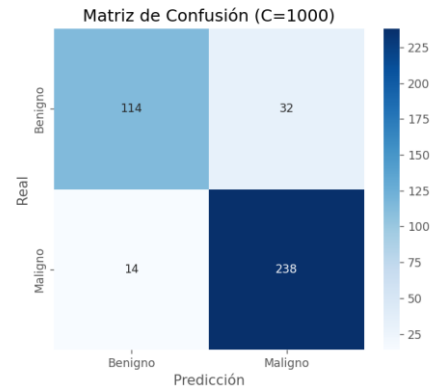
```



```

--- Evaluación del Modelo SVM con C=1000 ---
Accuracy: 88.44%
Precisión: 88.48%
Recall: 88.44%
Matriz de Confusión:
[[114 32]
 [ 14 238]]

```

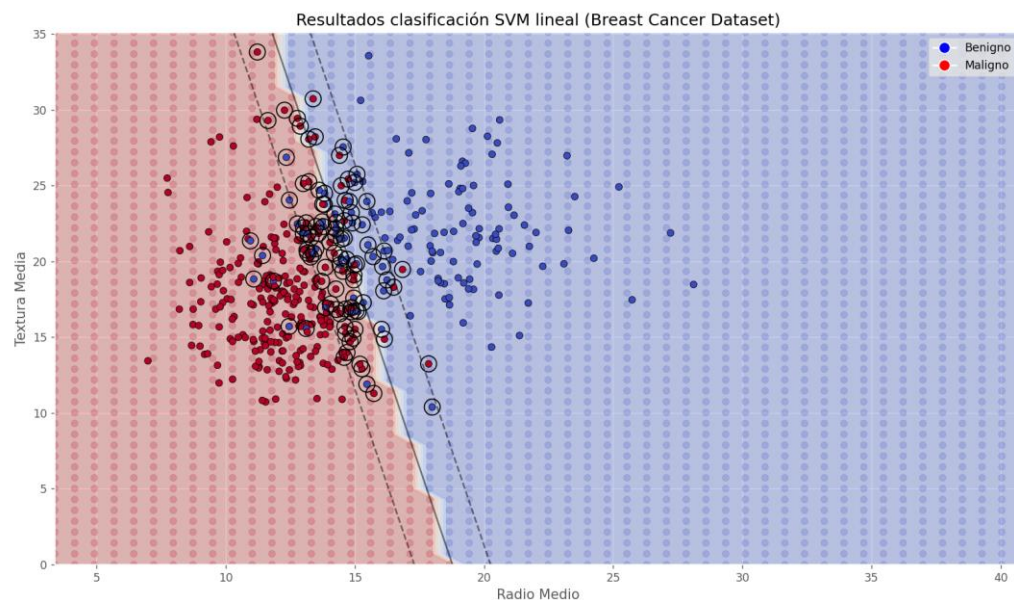
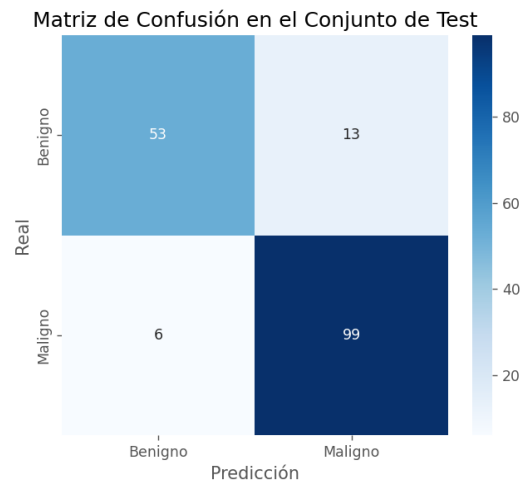


Mejor parámetro C encontrado: 100

```

--- Evaluación final en el conjunto de test ---
Accuracy: 88.89%
Precisión: 88.95%
Recall: 88.89%
Matriz de Confusión (test):
[[53 13]
 [ 6 99]]

```



Conclusiones

Al finalizar la práctica de clasificación utilizando Support Vector Machines (SVM) en Python, se pudo constatar que este algoritmo es una poderosa herramienta para resolver problemas de clasificación binaria. A lo largo del proceso, se resaltó la importancia de seleccionar y ajustar correctamente los hiperparámetros, en particular el parámetro C , para optimizar el modelo y maximizar el margen de separación entre las clases.

Durante el desarrollo del modelo, se subrayó la relevancia de una adecuada preparación de los datos, como la correcta selección de características, lo cual influye directamente en el rendimiento de SVM. Además, el uso de técnicas de validación cruzada con k-folds permitió evaluar el rendimiento del modelo en diferentes subconjuntos de datos, lo que ayuda a prevenir el sobreajuste y mejorar su capacidad de generalización. El análisis del desempeño se complementó con el cálculo de métricas clave como la exactitud (accuracy), la precisión (precision) y el recall, que ofrecieron una visión detallada de la capacidad del modelo para clasificar correctamente las instancias del dataset.

Asimismo, la visualización gráfica de los límites de decisión y de la matriz de confusión proporcionó una comprensión más profunda de cómo el modelo clasifica los datos y dónde se producen los errores. Esta representación fue particularmente útil para observar los vectores de soporte, que determinan el hiperplano de separación.

En resumen, esta práctica ha demostrado que SVM no solo es eficaz en la clasificación binaria, sino que también puede beneficiarse de una cuidadosa sintonización de hiperparámetros y visualización de resultados. Además, ha reforzado la importancia de herramientas de Python como Numpy, Seaborn y Matplotlib para el desarrollo de modelos en el ámbito de la ciencia de datos y el aprendizaje máquina. La experiencia adquirida ha sido fundamental para profundizar en el aprendizaje automático, brindando una base sólida para futuros proyectos en este campo.

Referencias

- InteractiveChaos. (s.f.). *Ventajas y desventajas de SVM*. InteractiveChaos. Recuperado el 10 de Octubre de 2024 de: <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/ventajas-y-desventajas-de-svm>
- MathWorks. (s.f.). *Introducción a Support Vector Machine (SVM)*. MathWorks. Recuperado el 10 de Octubre de 2024 de: <https://la.mathworks.com/discovery/support-vector-machine.html>