



Proyecto Tercer Parcial – Clasificación

Nombre: Odalys Yamilet Piemntel Juárez

Materia: Introducción al aprendizaje máquina

Fecha: 13 de Junio de 2025

1. 📁 Selección de base de datos

- Base de datos seleccionada: **load_breast_cancer**
- Número de columnas: 31
- Descripción general del dataset:
 - El conjunto de datos contiene características calculadas a partir de imágenes digitalizadas de la masa mamaria, que describen las características de los núcleos celulares presentes en la imagen. El objetivo es predecir si un tumor es maligno o benigno.

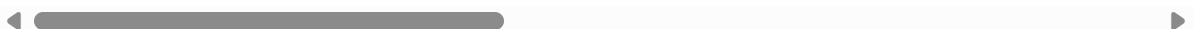
In [108...

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
data = load_breast_cancer()
# Cargar el dataset
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[108...

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	symr
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	C
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	C
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	C
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	C
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	C

5 rows × 31 columns



In [109...

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   mean radius                          569 non-null    float64
1   mean texture                         569 non-null    float64
2   mean perimeter                      569 non-null    float64
3   mean area                           569 non-null    float64
4   mean smoothness                     569 non-null    float64
5   mean compactness                    569 non-null    float64
6   mean concavity                      569 non-null    float64
7   mean concave points                 569 non-null    float64
8   mean symmetry                       569 non-null    float64
9   mean fractal dimension              569 non-null    float64
10  radius error                        569 non-null    float64
11  texture error                       569 non-null    float64
12  perimeter error                     569 non-null    float64
13  area error                          569 non-null    float64
14  smoothness error                    569 non-null    float64
15  compactness error                   569 non-null    float64
16  concavity error                     569 non-null    float64
17  concave points error                569 non-null    float64
18  symmetry error                      569 non-null    float64
19  fractal dimension error             569 non-null    float64
20  worst radius                        569 non-null    float64
21  worst texture                       569 non-null    float64
22  worst perimeter                     569 non-null    float64
23  worst area                          569 non-null    float64
24  worst smoothness                    569 non-null    float64
25  worst compactness                   569 non-null    float64
26  worst concavity                     569 non-null    float64
27  worst concave points                569 non-null    float64
28  worst symmetry                      569 non-null    float64
29  worst fractal dimension              569 non-null    float64
30  target                             569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

```
In [110... df=df.dropna() #elimina los datos nulos
```

2. Definición de variable objetivo

- **Variable objetivo (target):** target
- **Tipo de variable objetivo:** Binaria
- **Variables descriptivas (features):**
-Todas, menos target

```
In [111... X=df.drop(["target"],axis=1)
y=df["target"]
```

3. División de datos en entrenamiento y prueba

- **Porcentaje de entrenamiento:** 70%
- **Porcentaje de prueba:** 30%

```
In [112...] from sklearn.model_selection import train_test_split
```

```
In [113...] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=10)
```

4. 🤖 Entrenamiento de modelos

◆ Modelo 1: Perceptrón

```
In [114...] from sklearn.linear_model import Perceptron
```

```
In [115...] perceptron=Perceptron(max_iter=1000,eta0=0.1,random_state=10)
```

```
In [116...] perceptron.fit(X_train,y_train)
```

```
Out[116...]
Perceptron
Perceptron(eta0=0.1, random_state=10)
```

```
In [117...] perceptron.score(X_test,y_test)
```

```
Out[117...] 0.8947368421052632
```

```
In [118...] y_pred_perceptron=perceptron.predict(X_test)
```

◆ Modelo 2: Bayes Ingenuo

- **Tipo usado:** GaussianNB (pero se probaron todos)
- **Justificación:** El clasificador Naive Bayes de tipo Gaussiano predice una variable binaria a partir de variables continuas, que son exactamente el tipo de datos que maneja nuestro dataset.

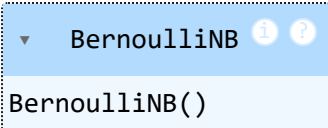
```
In [119...] from sklearn.naive_bayes import GaussianNB,BernoulliNB,MultinomialNB
```

```
In [120...] df["target"].unique()
```

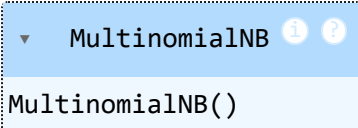
```
Out[120...] array([0, 1])
```

```
In [121...] gaus=GaussianNB()
ber=BernoulliNB()
multi=MultinomialNB()
```

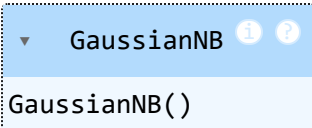
```
In [122...] ber.fit(X_train,y_train)
```

Out[122...  BernoulliNB()
BernoulliNB()

In [123... `multi.fit(X_train,y_train)`

Out[123...  MultinomialNB()
MultinomialNB()

In [124... `gaus.fit(X_train,y_train)`

Out[124...  GaussianNB()
GaussianNB()

In [125... `ber.score(X_test,y_test)`

Out[125... 0.6549707602339181

In [126... `multi.score(X_test,y_test)`

Out[126... 0.9239766081871345

In [127... `gaus.score(X_test,y_test)`

Out[127... 0.9590643274853801

In [128... `y_pred_gaussNB=gaus.predict(X_test)`

◆ Modelo 3: Máquina de Soporte Vectorial (SVM)

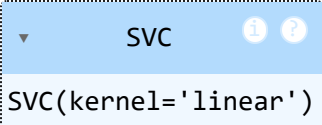
- **Kernel utilizado:** Linear / RBF / Linear /Poly
- **¿Escalaste los datos?:** Sí

In [129... `from sklearn.svm import SVC`
`from sklearn.preprocessing import StandardScaler`

In [130... `scaler = StandardScaler()`
`X_train_scaled = scaler.fit_transform(X_train)` # Ajusta y transforma el entrenamiento
`X_test_scaled = scaler.transform(X_test)`

In [131... `svm_lineal=SVC(kernel='linear')`

In [132... `svm_lineal.fit(X_train_scaled,y_train)`

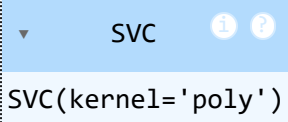
Out[132...  SVC(kernel='linear')

In [133... `svm_lineal.score(X_test_scaled,y_test)`

Out[133... 0.9532163742690059

In [134... `svm_poly=SVC(kernel='poly',degree=3)`

In [135... `svm_poly.fit(X_train_scaled,y_train)`

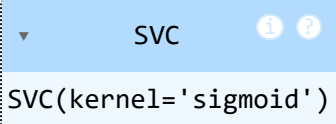
Out[135...  SVC(kernel='poly')

In [136... `svm_poly.score(X_test_scaled,y_test)`

Out[136... 0.8947368421052632

In [137... `svm_sigmoide=SVC(kernel='sigmoid')`

In [138... `svm_sigmoide.fit(X_train_scaled,y_train)`

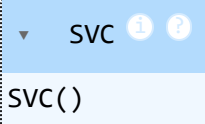
Out[138...  SVC(kernel='sigmoid')

In [139... `svm_sigmoide.score(X_test_scaled,y_test)`

Out[139... 0.9590643274853801

In [140... `svm_rbf=SVC(kernel='rbf') #radial basis function`

In [141... `svm_rbf.fit(X_train_scaled,y_train)`

Out[141...  SVC()

In [142... `svm_rbf.score(X_test_scaled,y_test)`

Out[142... 0.9824561403508771

In [143... `y_pred_svm=svm_rbf.predict(X_test_scaled)`

El hecho de que el kernel RBF haya ofrecido la mejor precisión (0.98) sugiere fuertemente que la relación entre las características y las etiquetas en el dataset load_breast_cancer es no

lineal y compleja, y que el kernel RBF fue el más adecuado para capturar esa complejidad sin sobreajustar los datos. Los otros kernels no lograron modelar esa no linealidad con la misma eficacia, ya sea porque eran demasiado simples (lineal), no tenían el grado polinomial adecuado, o no eran tan robustos para este tipo de datos (sigmoide).

5. Resultados y comparación

Métricas obtenidas

- Accuracy
- Matriz de confusión
- Reporte de clasificación

◆ Modelo 1: Perceptrón

```
In [144... from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

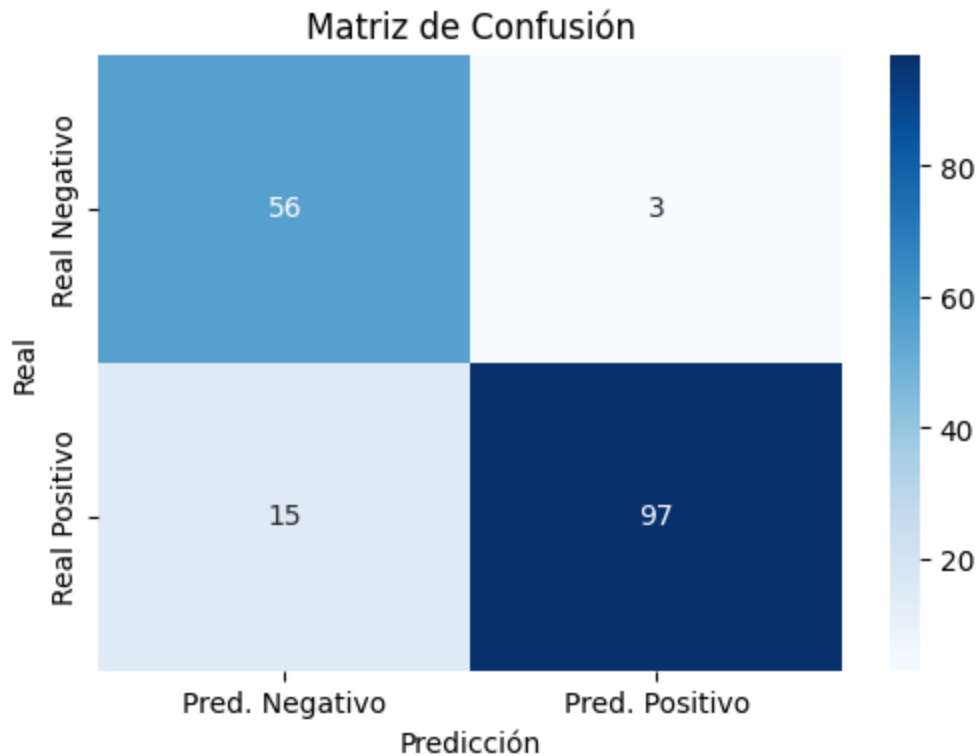
```
In [146... # 1. Accuracy
accuracy = accuracy_score(y_test, y_pred_perceptron)
print(f"📊 Accuracy: {accuracy:.4f}")

# 2. Matriz de confusión
cm = confusion_matrix(y_test, y_pred_perceptron)

# Visualizar matriz de confusión
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Pred. Negativo', 'Pred. Positivo'],
            yticklabels=['Real Negativo', 'Real Positivo'])
plt.title("Matriz de Confusión")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()

# 3. Reporte de clasificación
print("📋 Reporte de Clasificación:\n")
print(classification_report(y_test, y_pred_perceptron, target_names=['Clase 0', 'Clase 1']))
```

```
📊 Accuracy: 0.8947
```



Reporte de Clasificación:

	precision	recall	f1-score	support
Clase 0	0.79	0.95	0.86	59
Clase 1	0.97	0.87	0.92	112
accuracy			0.89	171
macro avg	0.88	0.91	0.89	171
weighted avg	0.91	0.89	0.90	171

◆ Modelo 2: Bayes Ingenuo

In [147...

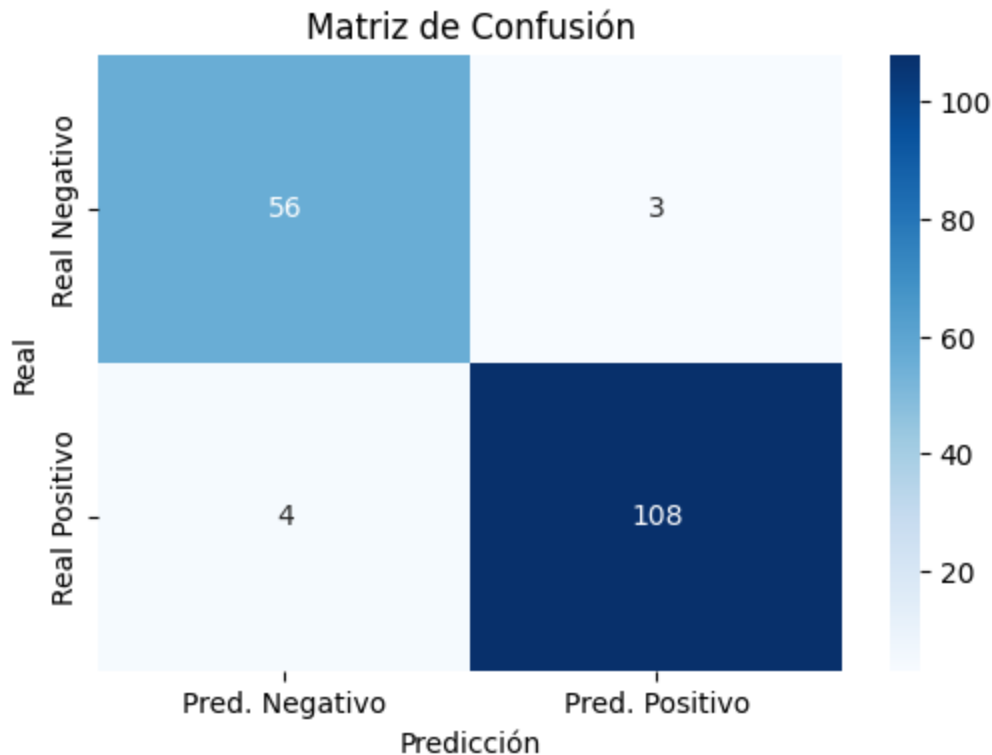
```
# 1. Accuracy
accuracy = accuracy_score(y_test, y_pred_gaussNB)
print(f"Accuracy: {accuracy:.4f}")

# 2. Matriz de confusión
cm = confusion_matrix(y_test, y_pred_gaussNB)

# Visualizar matriz de confusión
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Pred. Negativo', 'Pred. Positivo'],
            yticklabels=['Real Negativo', 'Real Positivo'])
plt.title("Matriz de Confusión")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()
```

```
# 3. Reporte de clasificación
print("📄 Reporte de Clasificación:\n")
print(classification_report(y_test, y_pred_gaussNB, target_names=['Clase 0', 'Clase 1']))
```

Accuracy: 0.9591



📄 Reporte de Clasificación:

	precision	recall	f1-score	support
Clase 0	0.93	0.95	0.94	59
Clase 1	0.97	0.96	0.97	112
accuracy			0.96	171
macro avg	0.95	0.96	0.95	171
weighted avg	0.96	0.96	0.96	171

◆ Modelo 3: Máquina de Soporte Vectorial (SVM)

In [148...

```
# 1. Accuracy
accuracy = accuracy_score(y_test, y_pred_svm)
print(f"📄 Accuracy: {accuracy:.4f}")

# 2. Matriz de confusión
cm = confusion_matrix(y_test, y_pred_svm)

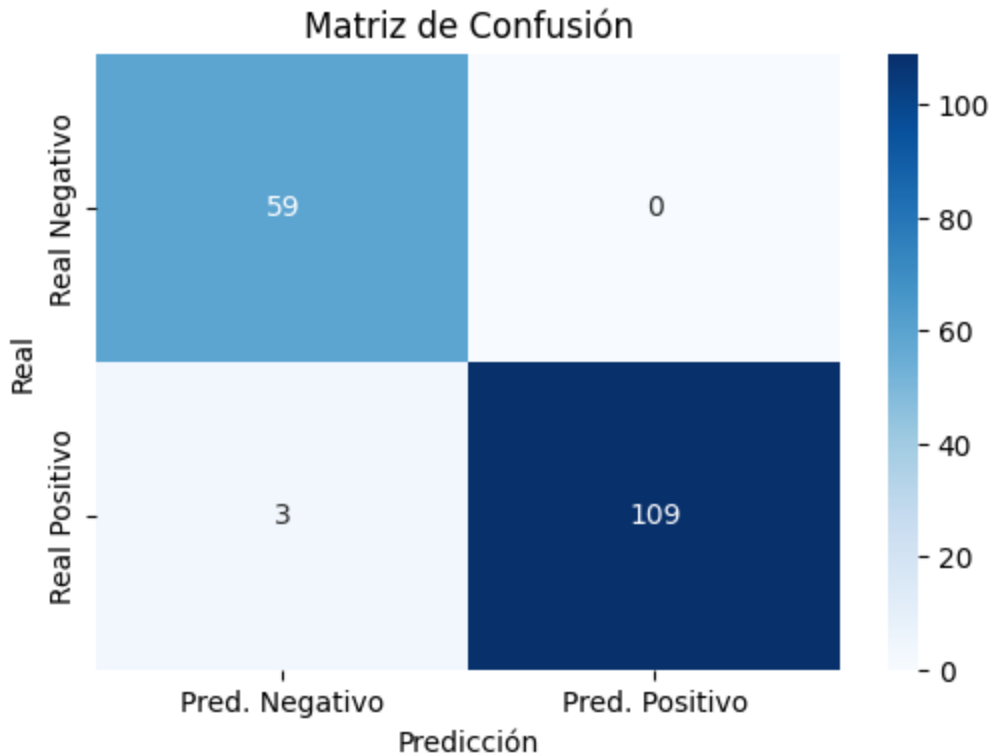
# Visualizar matriz de confusión
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Pred. Negativo', 'Pred. Positivo'],
            yticklabels=['Real Negativo', 'Real Positivo'])
```



```
plt.title("Matriz de Confusión")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()

# 3. Reporte de clasificación
print("📄 Reporte de Clasificación:\n")
print(classification_report(y_test, y_pred_svm, target_names=['Clase 0', 'Clase 1'])
```

34 Accuracy: 0.9825



📄 Reporte de Clasificación:

	precision	recall	f1-score	support
Clase 0	0.95	1.00	0.98	59
Clase 1	1.00	0.97	0.99	112
accuracy			0.98	171
macro avg	0.98	0.99	0.98	171
weighted avg	0.98	0.98	0.98	171

Comparación de modelos

Modelo	Accuracy	Observaciones breves
Perceptrón	0.89%	Buen inicio, pero limitado en datos no lineales; alta tasa de falsos negativos para Clase 1
Naive Bayes	0.96%	Rápido y efectivo, con buen balance; asume independencia de características, lo que puede ser una limitación

Modelo	Accuracy	Observaciones breves
SVM	0.98%	Excelente precisión, robusto con no linealidades (kernel RBF); cero falsos positivos

6. Conclusiones

- ¿Cuál modelo fue el más eficaz y por qué?

SVM con un kernel RBF (Radial Basis Function) fue mejor que Perceptron y Bayes Gaussiano por varias razones, las cuales se reflejan en las métricas:

-Perceptron: Es un modelo lineal. En los resultados, el Perceptron tiene un número significativamente alto de Falsos Negativos (15), lo que indica que no pudo identificar correctamente a un número considerable de casos positivos.

-Bayes Gaussiano: Aunque es un clasificador probabilístico que asume una distribución gaussiana para las características de cada clase, puede tener limitaciones si las distribuciones reales son más complejas o si hay una superposición significativa entre ellas que no se ajusta bien a las suposiciones gaussianas. En los resultados, tiene algunos Falsos Positivos y Falsos Negativos.

-SVM (RBF kernel): El kernel RBF permite a SVM transformar los datos a un espacio de mayor dimensión donde pueden ser linealmente separables, incluso si no lo eran en el espacio original. Esto lo hace muy potente para manejar relaciones no lineales y complejas en los datos. El hecho de que tenga cero Falsos Positivos y solo tres Falsos Negativos demuestra su capacidad superior para aprender la verdadera separación entre las clases en este dataset.

-Tolerancia a Outliers (con Parámetros de Regularización): El kernel RBF en SVM (con una buena selección de hiperparámetros) puede ser muy efectivo para manejar datos ruidosos o con outliers, ajustando la complejidad del modelo para evitar el sobreajuste (overfitting).

- ¿Qué dificultades encontraste?
 - Durante el desarrollo de este proyecto, mis principales desafíos no fueron de índole técnica o de implementación con las librerías. Más bien, la dificultad radicó en la curiosidad y la necesidad de experimentar con la optimización de los modelos. Esto implicó una exploración sistemática de diferentes parámetros y configuraciones para ajustar cada algoritmo y maximizar su rendimiento.
 - En el ámbito teórico, mi mayor reto inicial fue comprender a fondo las diferencias y aplicaciones específicas de los distintos tipos de kernels disponibles para las

Máquinas de Vectores de Soporte (SVM). Aunque al principio generó algunas dudas, esta incertidumbre se resolvió a través de una investigación activa y la aplicación práctica, lo que me permitió entender cómo cada kernel (lineal, polinomial, RBF, sigmoide) afecta la capacidad del modelo para modelar relaciones lineales y no lineales en los datos.

- ¿Qué aprendiste sobre clasificación en este proyecto?
 - En este proyecto, aprendí que la elección del modelo de clasificación es crucial y altamente dependiente del dataset en cuestión. Aunque los algoritmos como SVM (Support Vector Machine) a menudo demuestran un rendimiento superior, no existe un modelo 'universalmente' mejor para todos los escenarios. Específicamente, este proyecto me permitió:
 - Comprender el uso y las características de los clasificadores de Bayes y cómo sus supuestos de independencia afectan su rendimiento.
 - Distinguir las capacidades del Perceptron como un clasificador lineal básico.
 - Explorar la versatilidad de las Máquinas de Vectores de Soporte (SVM), especialmente la importancia de los distintos kernels (como el RBF, lineal, polinomial y sigmoide) para manejar la linealidad y no linealidad de los datos.
 - Evaluar y comparar el rendimiento de estos tres modelos (Perceptron, Bayes, SVM) para identificar cuál se ajusta mejor a las características del conjunto de datos `load_breast_cancer`.