Guía de estudio -Aprendizaje Supervisado: Clasificación II

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

Esta guía de continuación se enfoca en dos poderosos algoritmos de clasificación ampliamente utilizados: la regresión logística y el Support Vector Machine (SVM). Además, veremos métodos de evaluación como el Área Bajo la Curva ROC (AUC). Comenzaremos con una revisión detallada de la regresión logística, un modelo lineal que realiza clasificación binaria y puede extenderse a clasificación multiclase. Explicaremos la teoría detrás de este algoritmo, su función de activación sigmoide y cómo encontrar los parámetros óptimos utilizando el método de máxima verosimilitud.

Posteriormente, nos adentraremos en el poderoso mundo de los Support Vector Machines (SVM). Estos algoritmos se destacan por su capacidad para realizar clasificaciones tanto lineales como no lineales, mediante la transformación de los datos a un espacio de mayor dimensión utilizando kernels. Analizaremos los hiper parámetros más relevantes, como C y gamma, y veremos cómo optimizarlos para mejorar el rendimiento del modelo.

Finalmente, exploramos una métrica fundamental para evaluar el rendimiento de los modelos de clasificación: el Área Bajo la Curva ROC (AUC). Aprenderemos en qué consiste la curva ROC, cómo interpretarla y cómo calcular el AUC. También veremos cómo utilizar el AUC para comparar diferentes modelos y seleccionar el más adecuado para un problema específico.

A lo largo de esta guía, combinaremos teoría y práctica con ejemplos en Python que nos permitirán comprender y aplicar de manera efectiva cada uno de estos algoritmos. Al finalizar, tendrás una sólida comprensión de la regresión logística, el SVM y el Área Bajo la Curva ROC, lo que te proporcionará herramientas poderosas para enfrentar problemas de clasificación en tus propios proyectos de machine learning.

¡Comencemos este emocionante viaje hacia el fascinante mundo de los modelos de aprendizaje supervisado para clasificación y la evaluación de su rendimiento mediante el Área Bajo la Curva ROC!

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Overfitting y Underfitting

El sobreajuste (overfitting) y el subajuste (underfitting) son dos problemas comunes que pueden ocurrir al entrenar modelos de aprendizaje supervisado. Estos fenómenos pueden afectar negativamente el rendimiento del modelo y su capacidad para generalizar en datos no vistos. Comprender estos conceptos es esencial para desarrollar modelos más robustos y efectivos.

Sobreajuste (Overfitting)

El sobreajuste ocurre cuando un modelo se ajusta excesivamente a los datos de entrenamiento. En otras palabras, el modelo memoriza los detalles y el ruido de los datos en lugar de aprender patrones generales que puedan aplicarse a nuevos datos. Como resultado, el modelo tiene un rendimiento excelente en el conjunto de entrenamiento pero se desempeña mal en datos no vistos (conjunto de prueba o datos en producción). El sobreajuste puede llevar a una alta varianza en el modelo.

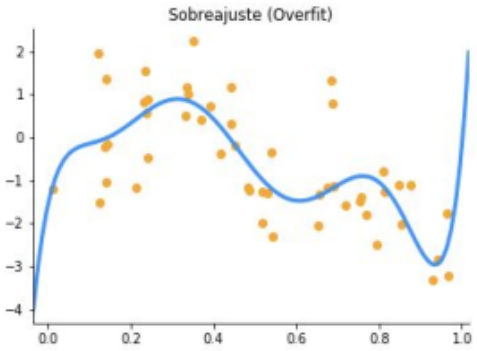

Imagen 1. Sobreajuste
Fuente: Desafío Latam

¿Cómo identificar el sobreajuste?

1. El rendimiento del modelo en el conjunto de entrenamiento es mucho mejor que en el conjunto de prueba.
2. La curva de aprendizaje muestra que el error en el conjunto de entrenamiento disminuye, pero el error en el conjunto de prueba se estabiliza o aumenta a medida que se incrementa el tamaño del conjunto de entrenamiento.

¿Cómo evitar el sobreajuste?

1. Utilizar más datos de entrenamiento si es posible. Un conjunto de entrenamiento más grande puede ayudar al modelo a capturar patrones más generales y reducir el riesgo de sobreajuste.

2. Simplificar el modelo: Reducir la complejidad del modelo, por ejemplo, disminuyendo el número de capas en una red neuronal o reduciendo la profundidad de un árbol de decisión.
3. Regularización: Introducir términos de penalización en la función de pérdida para controlar los coeficientes del modelo y evitar que se vuelvan demasiado grandes.
4. Validación cruzada: Utilizar técnicas de validación cruzada para evaluar el rendimiento del modelo y ajustar los hiper parámetros para evitar el sobreajuste.

Subajuste (Underfitting)

El subajuste ocurre cuando un modelo no es lo suficientemente complejo para capturar los patrones presentes en los datos. Como resultado, el modelo no se ajusta bien a los datos de entrenamiento y tiene un rendimiento deficiente tanto en el conjunto de entrenamiento como en el conjunto de prueba. El subajuste puede llevar a un alto sesgo en el modelo.

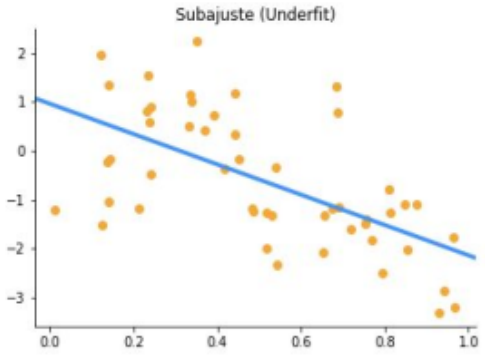

Imagen 2. Subajuste
Fuente: Desafío Latam

¿Cómo identificar el sobreajuste?

1. El rendimiento del modelo en el conjunto de entrenamiento y el conjunto de prueba es bajo.
2. La curva de aprendizaje muestra que el error en el conjunto de entrenamiento no disminuye significativamente a medida que se incrementa el tamaño del conjunto de entrenamiento.

¿Cómo evitar el sobreajuste?

1. Aumentar la complejidad del modelo: Considerar modelos más complejos que puedan capturar patrones más detallados en los datos.
2. Añadir más características: Si el modelo no tiene suficiente información para aprender, puede ser útil agregar más características relevantes.
3. Ajustar los hiperparámetros: Experimentar con diferentes valores de hiperparámetros para mejorar el rendimiento del modelo.

Regresión Logística

La regresión logística es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación binaria. Aunque su nombre contiene la palabra "regresión", es importante destacar que la regresión logística se utiliza para predecir la probabilidad de pertenecer a una de las dos clases (1 o 0), en lugar de predecir una variable continua.

El objetivo de la regresión logística es modelar la relación entre una o más variables predictoras (características) y la probabilidad de que un evento ocurra (probabilidad de pertenecer a una clase específica). Utiliza una función logística (también conocida como función sigmoide) para realizar esta modelización.

La función logística está definida como: $f(z) = \frac{1}{1+e^{-z}}$

Donde z es una combinación lineal de las variables predictoras y sus respectivos coeficientes.

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots\ldots + \beta_n x_n$$

El modelo de regresión logística asume que la relación entre las variables predictoras y la probabilidad de pertenecer a la clase positiva sigue una forma logística, lo que permite que la probabilidad esté en el rango de 0 a 1.

La función logística transforma la combinación lineal z en una probabilidad P de que la variable objetivo Y pertenezca a la clase positiva (1):

$$P(Y = 1|X) = f(z)$$
$$P(Y = 0|X) = 1 - P(Y = 1|X)$$

En la regresión logística, el objetivo es encontrar los valores óptimos de los coeficientes β que maximizan la verosimilitud de los datos observados. Esto se logra mediante la técnica de máxima verosimilitud, que busca maximizar la probabilidad conjunta de observar los datos reales bajo el modelo de regresión logística. En términos más simples, el algoritmo busca los valores de los coeficientes β que hacen que el modelo prediga con mayor precisión las clases observadas en el conjunto de entrenamiento.

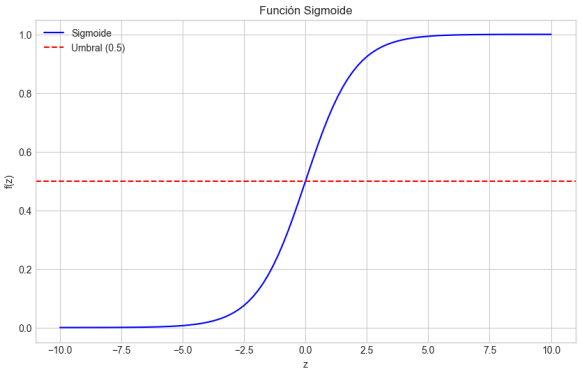

Imagen 3: Función Logística
Fuente: Desafío Latam

Ecuación de máxima verosimilitud

El método de máxima verosimilitud es una técnica estadística utilizada para estimar los parámetros desconocidos de un modelo estadístico basándose en los datos observados. Su objetivo es encontrar los valores de los parámetros que maximicen la probabilidad de observar los datos reales, asumiendo que el modelo propuesto es la verdadera distribución subyacente de los datos.

En el contexto de la regresión logística, por ejemplo, el método de máxima verosimilitud busca encontrar los coeficientes óptimos que hacen que el modelo de regresión logística prediga con la mayor probabilidad posible las clases observadas en el conjunto de entrenamiento.

Procedimiento

Supongamos que tenemos un conjunto de datos $\{(x_1, y_1), (x_2, y_2), \ldots\ldots, (x_n, y_n)\}$, donde x_1 representa las variables predictoras y y_i es la variable objetivo (0 o 1 en el caso de clasificación binaria).

1. **Función de probabilidad conjunta:** Para la regresión logística, asumimos que las variables objetivo y_i siguen una distribución de Bernoulli con una probabilidad $P(Y = 1|X) = p_i$ que es modelada por la función logística. La función de probabilidad conjunta para los datos observados se define como el producto de las probabilidades individuales para cada dato:

$$L(\beta) = \prod_{i=1}^{n} P(Y_i | X_i; \beta)$$

Donde β son los coeficientes que queremos encontrar.

2. **Función de verosimilitud:** La función de verosimilitud, $L(\beta)$, representa la probabilidad de observar los datos dados los valores de los parámetros β. Para

facilitar el cálculo, es más común trabajar con el logaritmo natural de la función de verosimilitud, conocido como log-verosimilitud:

$$log\ L(\beta) = \sum_{i=1}^{n} log\ P(Y_i | X_i; \beta)$$

3. **Estimación de parámetros:** El objetivo es encontrar los valores de los parámetros β que maximicen la función de verosimilitud, es decir, que aumenten la probabilidad de que los resultados sean los predichos. Pero en general los algoritmos son más eficientes al minimizar, por lo que buscaremos minimizar el negativo del log-verosimilitud (que suele llamarse función de pérdida o función de coste):

4. **Técnicas de optimización:** Para encontrar los valores óptimos de los parámetros β, se utilizan métodos de optimización numérica, como el descenso de gradiente o algoritmos más sofisticados, que ajustan gradualmente los valores de los coeficientes hasta encontrar un mínimo local de la función de pérdida.

5. **Resultados:** Una vez que se encuentran los valores óptimos de los coeficientes β, el modelo está listo para hacer predicciones sobre nuevos datos y clasificarlos en las clases correspondientes.

El método de máxima verosimilitud es ampliamente utilizado en estadística y machine learning debido a su propiedad de eficiencia asintótica y sus propiedades estadísticas deseables. Es una herramienta poderosa para la estimación de parámetros y permite obtener estimaciones óptimas basadas en los datos observados.

Ventajas y Desventajas

En general, la regresión logística es una técnica sólida para problemas de clasificación binaria y puede proporcionar resultados interpretables y probabilidades de clasificación. Sin embargo, es importante tener en cuenta sus limitaciones y considerar otros algoritmos cuando se enfrenten problemas más complejos o con más clases.

Ventajas

1. **Eficiente y fácil de implementar**: La regresión logística es un algoritmo relativamente simple y computacionalmente eficiente. Es fácil de implementar y entender, lo que lo hace una buena opción para problemas de clasificación binaria.

2. **Interpretación de coeficientes:** Los coeficientes estimados en la regresión logística proporcionan información sobre la influencia y dirección de cada variable predictora en la probabilidad de pertenecer a una clase. Esto permite una interpretación más clara de las relaciones entre las características y la variable objetivo.

3. **Modelo probabilístico**: La regresión logística proporciona probabilidades de clasificación en lugar de simplemente clases, lo que permite obtener información más detallada sobre las predicciones. Estas probabilidades pueden ser útiles en contextos donde se requiere información de confianza en las predicciones.

4. **Toma en cuenta valores atípicos:** A diferencia de algunos otros algoritmos, la regresión logística no se ve fuertemente afectada por valores atípicos o datos ruidosos.

5. **Flexibilidad en la selección de características**: Puede trabajar con una variedad de tipos de variables, incluidas variables categóricas y numéricas, y se puede utilizar para realizar selección de características mediante la inclusión o exclusión de variables predictivas.

Desventajas

1. **Limitado a problemas de clasificación binaria:** La regresión logística se limita a problemas de clasificación binaria, es decir, aquellos que tienen solo dos clases. Para problemas con más de dos clases, se deben utilizar otras técnicas, como la regresión logística multinomial o clasificadores multiclase.

2. **No es adecuado para relaciones no lineales complejas:** La regresión logística es un modelo lineal, lo que significa que no puede capturar relaciones no lineales complejas entre las variables predictoras y la variable objetivo. Para problemas con relaciones no lineales, se deben considerar otros algoritmos más complejos.

3. **Sensibilidad a características irrelevantes o colineales:** La presencia de características irrelevantes o altamente colineales puede afectar negativamente el rendimiento de la regresión logística, ya que puede introducir ruido en el modelo y afectar la interpretación de los coeficientes.

4. **Requiere una cantidad suficiente de datos**: La regresión logística puede requerir una cantidad suficiente de datos de entrenamiento para estimar los coeficientes de manera confiable y evitar el sobreajuste.

Actividad guiada: Probando la regresión logística

Importando librerías

Usamos las librerías clasicas que hemos visto hasta el momento:

- *numpy: para manejo de datos*
- *matplotlib: para generar gráficos*
- *sklearn.dataset: para cargar un dataset*
- *sklearn.model_selection: para dividir las muestras*
- *sklearn.preprocessing: Para procesar los datos*
- *sklearn.linear_model LogisticRegression: Para instanciar el modelo*

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

Carga de datos

```
# Cargar el conjunto de datos
data = load_breast_cancer()
X, y = data.data, data.target
```

Preprocesar los datos (Escalando)

```
# Escalar las características para que tengan media 0 y desviación estándar 1
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Seleccionar solo dos características para el plot
feature1_idx = 0
feature2_idx = 1
X_2d = X_scaled[:, [feature1_idx, feature2_idx]]
```

Dividir datos y entrenar modelo

```
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_2d, y, test_size=0.2,
random_state=42)
```

```
# Crear y entrenar el modelo de regresión logística
model = LogisticRegression()
model.fit(X_train, y_train)
```

Graficar los datos

```
# Obtener los coeficientes y el intercepto del modelo
coeficientes = model.coef_
intercepto = model.intercept_

# Crear un meshgrid para el plot
x_min, x_max = X_2d[:, 0].min() - 1, X_2d[:, 0].max() + 1
y_min, y_max = X_2d[:, 1].min() - 1, X_2d[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# Predecir las clases para cada punto del meshgrid
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot de las regiones de decisión y los puntos de entrenamiento
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.6, cmap=plt.cm.coolwarm)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.coolwarm,
edgecolors='k')
plt.xlabel(data.feature_names[feature1_idx])
plt.ylabel(data.feature_names[feature2_idx])
plt.title("Regresión Logística - Regiones de Decisión")
plt.colorbar(label="Clase")
plt.show()
```

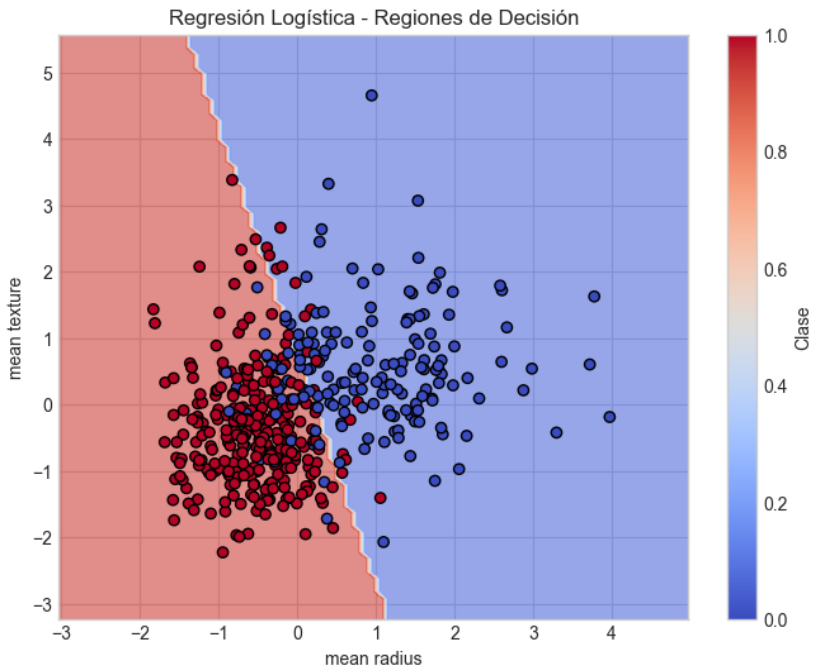

Imagen 4: Regiones de decisión
Fuente: Desafío Latam

Support Vector Machine (SVM)

El Support Vector Machine (SVM) es uno de los algoritmos más destacados en el campo del aprendizaje supervisado, reconocido por su eficacia en problemas de clasificación lineal y no lineal. Su popularidad se debe a su capacidad para encontrar soluciones óptimas en entornos de alta dimensionalidad y complejidad, lo que lo convierte en una herramienta valiosa para abordar una amplia gama de problemas del mundo real.

La idea central detrás de SVM radica en su enfoque en la maximización de la separabilidad entre diferentes clases en el espacio de características. El objetivo fundamental es encontrar el hiperplano óptimo que mejor separe las muestras de distintas clases, logrando así una clasificación precisa. Un hiperplano es simplemente un subespacio de dimensión (n-1) en un espacio de características n-dimensional, que actúa como una frontera de decisión para asignar nuevas muestras a una clase u otra.

¿Cómo se logra esta separabilidad óptima? SVM se basa en la idea de encontrar el hiperplano que maximice la distancia entre los puntos de datos más cercanos de diferentes clases, conocidos como vectores de soporte (support vectors). Estos puntos son fundamentales, ya que determinan la posición y orientación del hiperplano, y la optimización se centra en mantener un margen máximo entre ellos.

Además, SVM se caracteriza por su buen rendimiento en conjuntos de datos pequeños o ruidosos, y su capacidad para manejar conjuntos de alta dimensionalidad sin sufrir el problema de la maldición de la dimensionalidad.

En resumen, Support Vector Machine (SVM) es una poderosa técnica de aprendizaje supervisado que busca maximizar la separabilidad entre clases en un espacio de características mediante el uso de hiperplanos óptimos. Su flexibilidad para abordar problemas lineales y no lineales, junto con su capacidad para trabajar con conjuntos de datos de alta dimensionalidad, lo convierten en una herramienta esencial para resolver problemas de clasificación en diversas áreas, como visión por computadora, procesamiento de lenguaje natural, medicina, entre otros. Comprender la intuición detrás de SVM nos permitirá aprovechar su potencial y aplicarlo de manera efectiva en proyectos de aprendizaje automático.

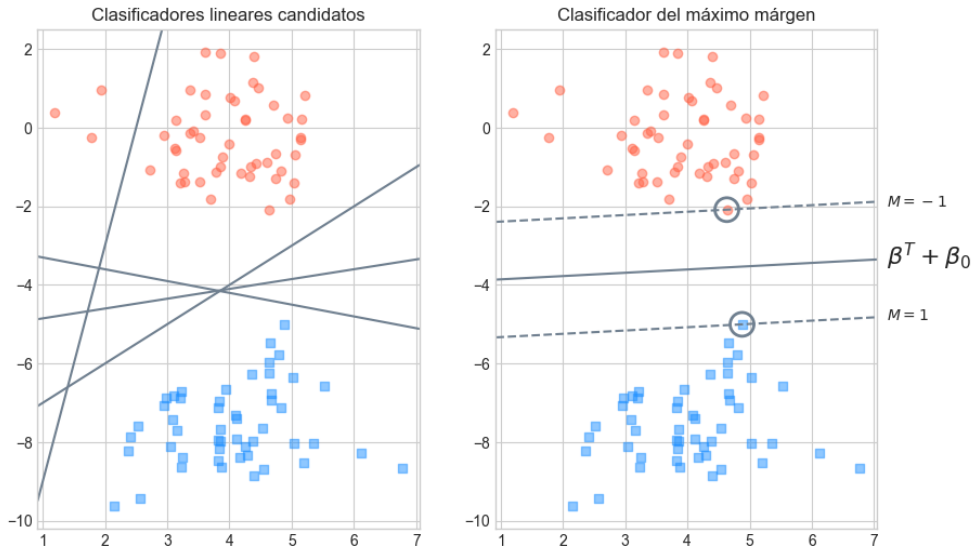

Imagen 5: Support Vector Machine
Fuente: Desafío Latam

Teoría de máquinas de soporte vectorial

La teoría detrás de Support Vector Machine (SVM) se basa en la idea de encontrar el hiperplano óptimo que maximice la separabilidad entre las muestras de diferentes clases en un espacio de características. El objetivo es lograr un modelo de clasificación que pueda generalizar bien a nuevos datos. A continuación, se presentan las principales fórmulas que sustentan el algoritmo SVM:

En un problema de clasificación binaria en un espacio de características n-dimensional, el hiperplano lineal es representado por una ecuación de la forma:

$$w^T * X + b = 0$$

Donde:

- w corresponde al vector de pesos o coeficientes del hiperplano.
- X corresponde al vector de características de una muestra.
- b corresponde al sesgo o término de sesgo del hiperplano.

Para clasificar una muestra x en una clase u otra, utilizamos la función de decisión $f(x)$, que se define como:

$$f(x) = w^T * x + b$$

- Si $f(x) \geq 0$, la muestra se clasifica como positiva.
- Si $f(x) < 0$, la muestra se clasifica como positiva.

Como se entrenan las máquinas de soporte vectorial

SVM busca encontrar el hiperplano óptimo que logre una separación máxima entre las muestras de diferentes clases. Para ello, introduce la noción de margen (γ), que es la distancia entre el hiperplano y los vectores de soporte (support vectors) más cercanos. Los vectores de soporte son aquellas muestras que están más cercanas al hiperplano y son fundamentales para la clasificación.

El margen (γ) se calcula como la distancia entre dos hiperplanos paralelos que pasan por los vectores de soporte más cercanos. La separación óptima es aquella que maximiza el margen señalado por las líneas punteadas.

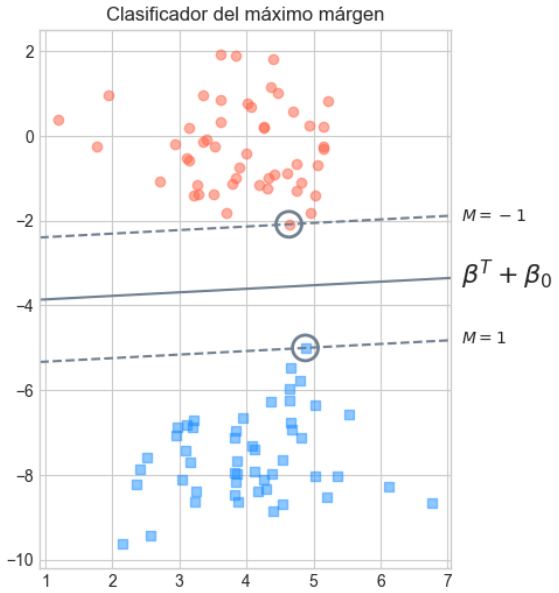

Imagen 6: Clasificador del máximo margen
Fuente: Desafío Latam

Hiperparametros

Kernel

En problemas de clasificación, es común que los datos no sean linealmente separables en el espacio de características original. Esto significa que no podemos encontrar un hiperplano que divida perfectamente las muestras de diferentes clases. Para abordar esta limitación, SVM emplea una técnica llamada kernel trick.

Los kernels son funciones que permiten transformar los datos del espacio de características original a un espacio de mayor dimensión donde se vuelven linealmente separables. Es decir, los kernels realizan una transformación no lineal de los datos, lo que facilita encontrar un hiperplano lineal en este nuevo espacio, incluso si los datos originales no son linealmente separables.

Dentro del contexto de SVM, la función de decisión lineal se define como:

$$f(x) = w^T * x + b$$

Donde w es el vector de pesos o coeficientes y b es el sesgo o término de sesgo del hiperplano.

Al introducir kernels, la función de decisión se modifica a:

$$f(x) = w^T * \Phi(x) + b$$

Donde $\Phi(x)$ es la transformación no lineal de la muestra x realizada por el kernel. En otras palabras, $\Phi(x)$ mapea la muestra x a un espacio de mayor dimensión donde se espera que los datos sean linealmente separables.

Existen diferentes tipos de kernels que se pueden utilizar en SVM, y cada uno tiene sus características y ventajas. Los kernels más comunes son:

1. **Kernel Lineal:** Es el kernel más simple y se utiliza para problemas linealmente separables. La función de mapeo $\Phi(x)$ simplemente es la identidad, lo que significa que no se realiza ninguna transformación.

2. **Kernel Polinómico:** Este kernel utiliza una función polinómica para realizar la transformación. La función de mapeo $\Phi(x)$ aumenta el espacio de características mediante polinomios de grado d.

3. **Kernel Radial (RBF)**: También conocido como kernel gaussiano, utiliza una función radial para realizar la transformación. La función de mapeo $\Phi(x)$ mapea las muestras a un espacio de características de infinitas dimensiones.

Los kernels son una poderosa herramienta en el contexto de SVM para abordar problemas de clasificación no lineales. Estas funciones de mapeo no lineal permiten transformar los

datos del espacio de características original a uno de mayor dimensión donde se vuelven linealmente separables, lo que amplía la capacidad de SVM para resolver una amplia variedad de problemas del mundo real. La elección del kernel adecuado es esencial para obtener un rendimiento óptimo en la clasificación de datos complejos y puede marcar la diferencia entre un modelo eficaz y uno que subutiliza su potencial.

Costo C

El parámetro o Costo C es uno de los hiperparámetros más importantes en SVM y se conoce como el parámetro de regularización. Su función es controlar el equilibrio entre la clasificación correcta de las muestras de entrenamiento y la complejidad del modelo. En otras palabras, C determina cuánto se penalizan los errores en la clasificación de las muestras de entrenamiento.

- **Cuando C es pequeño:** Se permite una mayor flexibilidad en el modelo, lo que significa que el algoritmo puede permitir un mayor número de errores en la clasificación de las muestras de entrenamiento. Esto puede resultar en un modelo con un margen más amplio y un mayor sesgo, lo que podría llevar a un subajuste (underfitting) en el conjunto de entrenamiento.

- **Cuando C es grande:** Se penalizan más los errores en la clasificación, lo que lleva a un modelo más rígido y ajustado a los datos de entrenamiento. Esto puede resultar en un modelo con un margen más estrecho y menor sesgo, lo que podría llevar a un sobreajuste (overfitting) en el conjunto de entrenamiento.

En general, el valor óptimo de C dependerá de la complejidad del problema y del conjunto de datos. Es común utilizar técnicas de validación cruzada para encontrar el valor de C que produzca el mejor rendimiento en datos no vistos y que generalice bien.

Gamma

El parámetro gamma es específico del kernel radial (RBF) y controla el alcance de influencia de un solo ejemplo de entrenamiento. En otras palabras, gamma determina cuánta influencia tiene un punto de entrenamiento sobre otros puntos en el espacio de características.

- **Cuando gamma es pequeño:** El alcance de influencia de un punto de entrenamiento es más amplio, lo que significa que los puntos se afectan entre sí en un área más grande del espacio de características. Esto puede resultar en un modelo con fronteras de decisión más suaves y más amplias, lo que podría llevar a un sobreajuste en el conjunto de entrenamiento.

- **Cuando gamma es grande:** El alcance de influencia de un punto de entrenamiento es más estrecho, lo que significa que los puntos afectan solo a puntos cercanos en el espacio de características. Esto puede resultar en un modelo con fronteras de decisión más complejas y ajustadas a los puntos de entrenamiento, lo que podría llevar a un subajuste en el conjunto de entrenamiento.

Al igual que con el parámetro C, encontrar el valor óptimo de gamma también depende de la naturaleza del problema y del conjunto de datos. Técnicas de validación cruzada pueden ayudar a determinar el mejor valor de gamma para un buen rendimiento general del modelo.

¡Manos a la obra! - Probando Hiperparámetros

Kernel

Importando librerías

Usamos las librerías clásicas que hemos visto hasta el momento:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.datasets import make_circles

plt.style.use('seaborn-whitegrid')
plt.rcParams['figure.figsize']=(10, 6)
```

Generar datos circulares

Generamos datos de clasificación con círculos concéntricos

```
X, y = make_circles(n_samples=300, noise=0.1, random_state=42, factor=0.5)
```

Probar kernel

Entrenamos modelos con kernel 'linear' y kernel 'rbf' y ploteamos los resultados.

```
plt.subplot(1,2, 1)
# Crear el modelo SVM con el kernel RBF (gaussiano)
model = SVC(kernel='linear', gamma=1)
# Entrenar el modelo
model.fit(X, y)
# Predecir las clases para todo el espacio de características
h = 0.02  # Tamaño del paso en el meshgrid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

# Plot del resultado
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolors='k')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.title('Support Vector Machine con Kernel Lineal')

plt.subplot(1,2, 2)
# Crear el modelo SVM con el kernel RBF (gaussiano)
model = SVC(kernel='rbf', gamma=1)
# Entrenar el modelo
model.fit(X, y)
# Predecir las clases para todo el espacio de características
h = 0.02  # Tamaño del paso en el meshgrid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

# Plot del resultado
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolors='k')
```

```
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.title('Support Vector Machine con Kernel RBF')
plt.show()
```

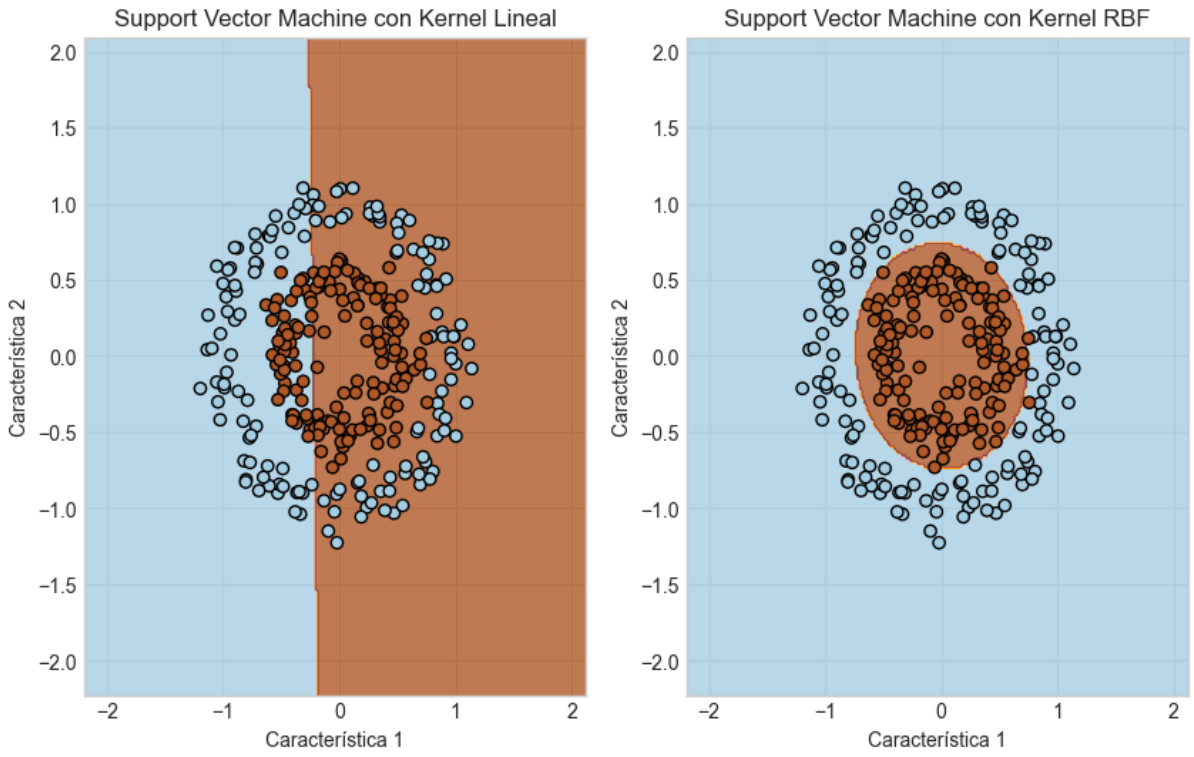

Imagen 7: Probando hiperparámetros
Fuente: Desafío Latam

Costo C

Generar datos XOR

Función para generar datos en forma de O exclusivo

```
fetch_lims = lambda x: [np.floor(np.min(x)), np.ceil(np.max(x))]

def get_joint_xy(xlim, ylim):
    x_mesh, y_mesh = np.meshgrid(
        np.linspace(xlim[0], xlim[1]),
        np.linspace(ylim[0], ylim[1])
    )
```

```
    joint_xy = np.vstack([
        x_mesh.ravel(),
        y_mesh.ravel()
    ]).T
    return x_mesh, y_mesh, joint_xy

def svm_logical_xor_data(nsize=400, random_state=11238):
    np.random.seed(random_state)
    x_xor = np.random.randn(nsize, 2)
    y_xor = np.logical_xor(x_xor[:, 0] > 0, x_xor[:, 1] > 0)
    y_xor = np.where(y_xor, 1, -1)
    return x_xor, y_xor
```

Probar diferentes valores

Probar con diferentes valores y generar gráficos de estos.

```
def svm_c_hyperparameter(X, y, c_range = [0.0001, 0.1, 1000]):
    get_xlim = fetch_lims(X[:, 0])
    get_ylim = fetch_lims(X[:, 1])
    x_mesh, y_mesh, joint_xy = get_joint_xy(get_xlim, get_ylim)

    for index, c in enumerate(c_range):
        if len(c_range) > 3:
            plt.subplot(2, 3, index + 1)
        else:
            plt.subplot(1, 3, index + 1)
        tmp_model = SVC(kernel = 'rbf', C=c, gamma=.01).fit(X, y)
        tmp_densities =
tmp_model.decision_function(joint_xy).reshape(x_mesh.shape)
        plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='orange',
alpha=.8, s=25, marker='s')
        plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='dodgerblue',
alpha=.8, s=25, marker='o')
        plt.contourf(x_mesh, y_mesh, tmp_densities, cmap='Greys', alpha=.5)
        plt.title("C: {}".format(c), fontsize=18)
        plt.tight_layout()

X, y = svm_logical_xor_data(nsize=75) # 1000,
svm_c_hyperparameter(X, y, c_range=[0.001, 0.1, 1, 100, 1000, 10000])
```

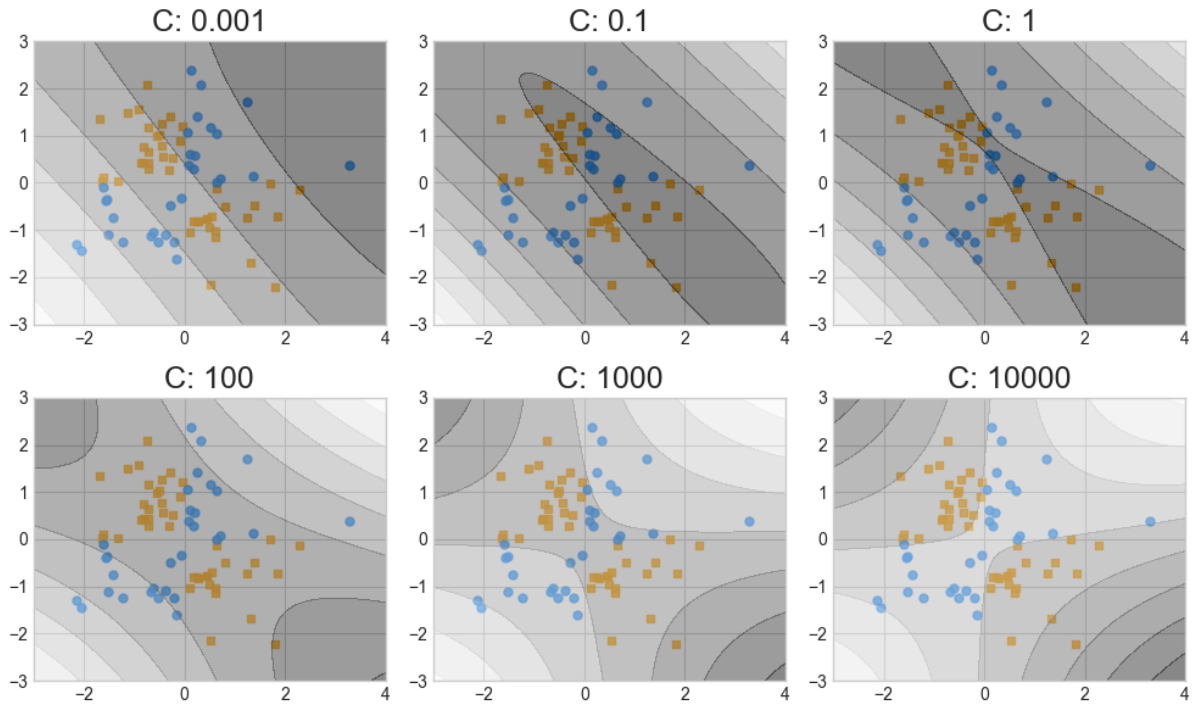

Imagen 8: Costo C
Fuente: Desafío Latam

Gamma

Probar diferentes datos de gamma

```
def svm_gamma_hyperparameter(X, y, gamma_range=[0.0001, 0.1, 1000]):
    get_xlim = fetch_lims(X[:, 0])
    get_ylim = fetch_lims(X[:, 1])
    x_mesh, y_mesh, joint_xy = get_joint_xy(get_xlim, get_ylim)

    for index, g in enumerate(gamma_range):
        if len(gamma_range) > 3:
            plt.subplot(2, 3, index + 1)
        else:
            plt.subplot(1, 3, index + 1)
        tmp_model = SVC(kernel = 'rbf', gamma=g, C=1).fit(X, y)
        tmp_densities =
tmp_model.decision_function(joint_xy).reshape(x_mesh.shape)
        plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='orange',
alpha=.8, s=25, marker='s')
        plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='dodgerblue',
alpha=.8, s=25, marker='o')
        plt.contourf(x_mesh, y_mesh, tmp_densities, cmap='Greys', alpha=.5)
        plt.title("Gamma: {}".format(g), fontsize=18)
```

```
        plt.tight_layout()

svm_gamma_hyperparameter(X, y, gamma_range=[0.0001, 0.01,0.1, 1, 10, 100])
```

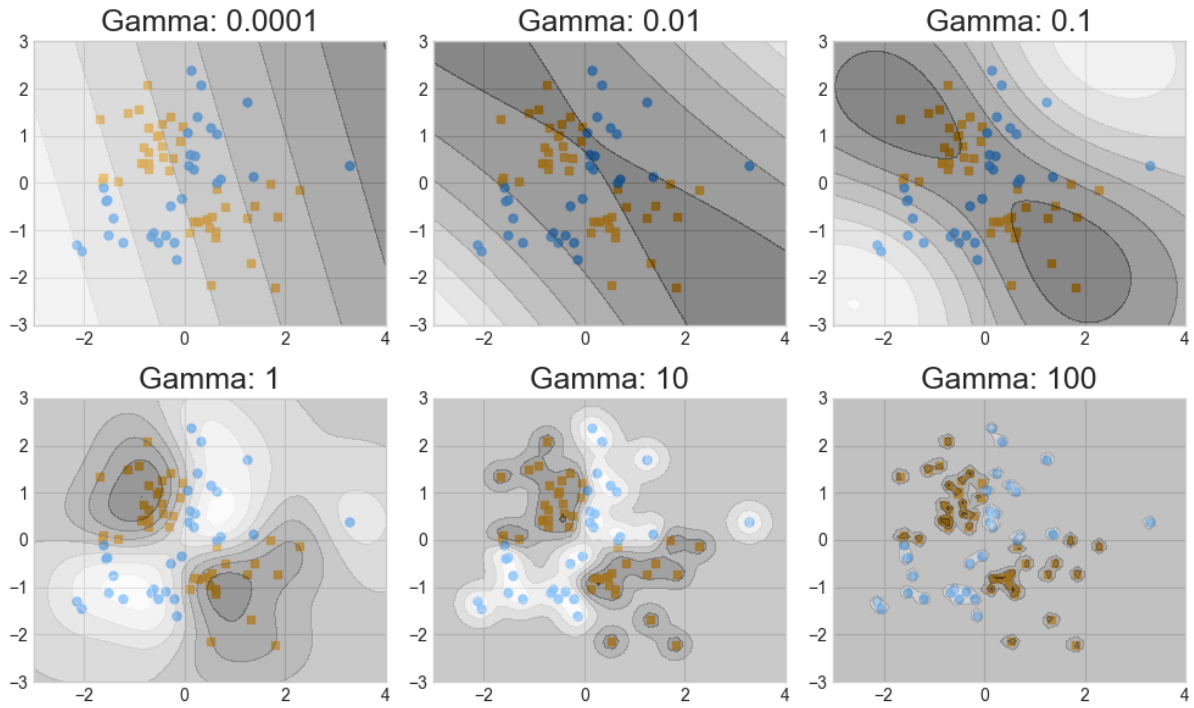

Imagen 9: Gamma
Fuente: Desafío Latam

Métricas de clasificación (ROC AUC)

La métrica AUC-ROC (Área Bajo la Curva de la Característica Operativa del Receptor) es una medida utilizada para evaluar el rendimiento de modelos de clasificación binaria. La curva ROC se construye trazando la tasa de verdaderos positivos (TPR) frente a la tasa de falsos positivos (FPR) para diferentes valores de umbral de clasificación.

Para entender cómo funciona la métrica AUC-ROC, es importante tener claros los conceptos de TPR y FPR:

- **TPR (True Positive Rate):** También conocida como Sensibilidad o Recall, mide la proporción de muestras positivas que se clasifican correctamente como positivas.
- **FPR (False Positive Rate):** Mide la proporción de muestras negativas que se clasifican incorrectamente como positivas.

La curva ROC representa la relación entre TPR y FPR al variar el umbral de clasificación del modelo. Al evaluar diferentes umbrales, obtenemos diferentes pares (TPR, FPR), lo que resulta en una curva en el espacio [0, 1] x [0, 1].

Interpretación

La interpretación de la curva ROC es sencilla: cuanto más cerca esté la curva del vértice superior izquierdo (coordenadas [0,1]), mejor será el rendimiento del modelo, ya que indica una alta tasa de verdaderos positivos y una baja tasa de falsos positivos. Un modelo con una curva ROC que se superponga con la línea diagonal (tasa de verdaderos positivos = tasa de falsos positivos) indica un rendimiento similar al azar, mientras que una curva ROC por debajo de la diagonal representa un rendimiento peor que el azar.

ROC AUC

El valor del AUC se calcula como el área bajo la curva ROC. El AUC oscila entre 0 y 1, donde un AUC de 0.5 indica que el modelo tiene un rendimiento similar al azar (clasificación aleatoria), y un AUC de 1.0 indica un rendimiento perfecto, es decir, el modelo clasifica todas las muestras correctamente.

La métrica AUC-ROC es independiente del punto de corte o umbral de clasificación. Esto significa que el AUC-ROC evalúa el rendimiento del modelo para todos los umbrales posibles, lo que proporciona una visión más completa de su capacidad de clasificación.

Es especialmente útil cuando se trabaja con conjuntos de datos desequilibrados, donde una clase tiene muchas más muestras que la otra. En tales casos, la precisión por sí sola puede ser engañosa, ya que un modelo que siempre predice la clase mayoritaria tendría una alta precisión, pero no sería útil en la práctica. La métrica AUC-ROC toma en cuenta tanto las tasas de verdaderos positivos como de falsos positivos, proporcionando una evaluación más equilibrada del rendimiento del modelo.

Comparar el AUC-ROC entre diferentes modelos de clasificación permite identificar cuál de ellos es mejor en términos de capacidad de discriminación y capacidad de clasificación correcta.

En resumen, la métrica AUC-ROC es una medida importante para evaluar el rendimiento de modelos de clasificación binaria. Su representación gráfica como la curva ROC ofrece información valiosa sobre la capacidad de clasificación del modelo en diferentes umbrales. Un AUC-ROC cercano a 1 indica un modelo altamente efectivo, mientras que un valor cercano a 0.5 sugiere un rendimiento similar al azar. El uso de la métrica AUC-ROC proporciona una evaluación más completa y equilibrada del rendimiento del modelo en comparación con otras métricas de clasificación.

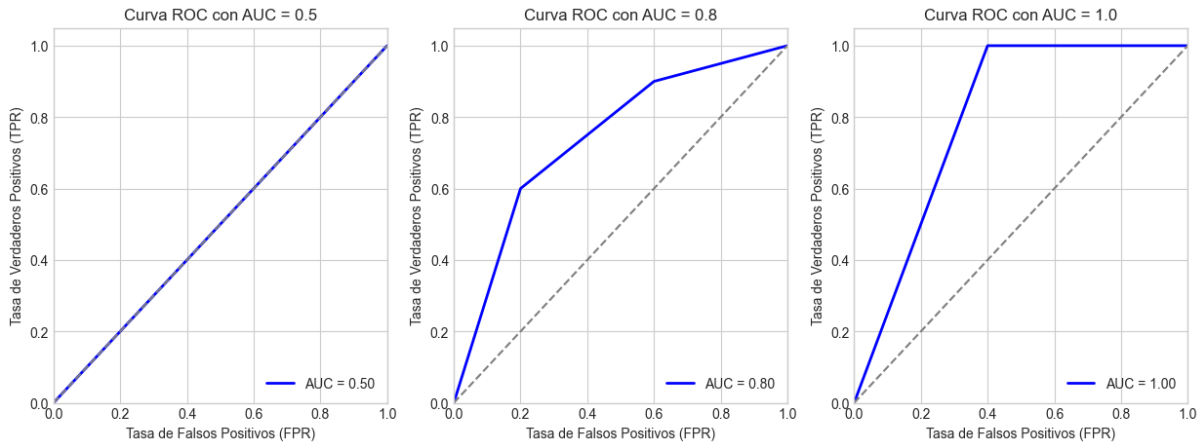

Imagen 10: ROC AUC
Fuente: Desafío Latam

Preguntas de proceso

Reflexiona:

- ¿Qué es el aprendizaje supervisado y en qué se diferencia de otros enfoques de aprendizaje automático?
- ¿Cómo se puede identificar y abordar el sobreajuste y subajuste?
- ¿Qué es la regresión logística y para qué se utiliza en el aprendizaje supervisado? Explique la función de decisión y cómo se realiza la clasificación en este algoritmo.
- ¿Cómo se pueden optimizar los hiperparámetros?
- ¿Qué es Support Vector Machine (SVM) y cómo funciona para resolver problemas de clasificación? Explique el concepto de hiperplano óptimo y los vectores de soporte.
- ¿Cómo afectan al rendimiento y la flexibilidad del modelo los hiperparámetros C y Gamma?
- ¿Cuál es la teoría detrás de la métrica ROC AUC y cómo se calcula? ¿Qué representa un valor de AUC-ROC igual a 0.5, 0.8 y 1.0?
- Explique cómo se utilizan los kernels en SVM y cómo ayudan a resolver problemas no lineales. ¿Cuál es la diferencia entre un kernel lineal, polinómico y radial (RBF)?
- ¿Qué ventajas y desventajas tienen la regresión logística y SVM como algoritmos de clasificación? ¿Cuándo sería apropiado usar uno sobre el otro?

Referencias bibliográficas

1. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
2. Python Data Science Handbook: https://jakevdp.github.io/PythonDataScienceHandbook
3. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

¡Continúa aprendiendo y practicando!