UNIVERSIDAD HEMISFERIOS

Unidad 2. Minería de Datos y Aprendizaje Automático

Todos los derechos reservados © Universidad Hemisferios

Introducción

En esta unidad se aplica algoritmos de aprendizaje automático supervisado (clasificación) sobre conjuntos de datos públicos siguiendo una metodología para el desarrollo de proyectos de análisis de datos y los aspectos éticos en el uso y aprovechamiento del big data.

Algunos algoritmos de clasificación clásicos son redes neuronales artificiales, vecinos más cercanos (K-NN), decisión tree y regresión logística para problemas de clasificación binaria y multi-clasificación, todos ellos utilizando el lenguaje Python y las herramientas Keras y TensorFlow.

Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

Objetivo

Aplicar algoritmos de aprendizaje automático (modelos predictivos) a conjunto de datos estructurados para el descubrimiento de patrones y la toma de decisiones utilizando Python, Keras y TensorFlow.

Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

2.1 Introducción al Aprendizaje Supervisado y no Supervisado

El aprendizaje supervisado y no supervisado son dos enfoques fundamentales en el campo del aprendizaje automático.

En el aprendizaje supervisado, se utilizan datos de entrenamiento que están etiquetados, lo que significa que cada ejemplo en el conjunto de datos tiene una etiqueta o respuesta conocida.

El aprendizaje supervisado se utiliza típicamente para desarrollar modelos predictivos. Esto incluye modelos de clasificación (como predecir si un correo electrónico es spam o no) y modelos de regresión (como predecir el precio de una casa en función de sus características).

Todos los derechos reservados © Universidad Hemisferios

En el aprendizaje no supervisado, los datos de entrenamiento no están etiquetados. El modelo debe encontrar patrones, estructuras o agrupamientos inherentes en los datos sin conocer las respuestas correctas de antemano.

El aprendizaje no supervisado se utiliza para técnicas descriptivas que ayudan a entender mejor la estructura y las relaciones en los datos. Esto incluye el clustering (para agrupar datos similares) y la reducción de dimensionalidad (para representar datos en un espacio de menor dimensión).

Ambos enfoques, supervisado y no supervisado, son fundamentales en la ciencia de datos y el aprendizaje automático, y se utilizan en una variedad de aplicaciones según los objetivos de modelado y los datos disponibles.

Todos los derechos reservados © Universidad Hemisferios

Técnicas de análisis de datos masivos:

Técnicas Predictivas:

Clasificación: El atributo a predecir (target) es de tipo cualitativo (categoría). ej. clasificación de objetos en imágenes: perro, gato, avión, laptop. Algoritmos: RNA (MLP), SVM, decision tree, K-NN, Naive Bayes, logistic regression, Random Forest (ensemble).

Regresión: Similar a la clasificación, pero el atributo a predecir es de tipo cuantitativo. Algoritmos: linear and multiple regression with least-squares method, RNA, Regression tree, SVM, Random Forest.

Todos los derechos reservados © Universidad Hemisferios

Técnicas Descriptivas:

Agrupamiento: Segmentación en grupos homogéneos. Ej. Clientes A, B, C. Algoritmos: k-means, Expectation Maximization (EM), K-NN.

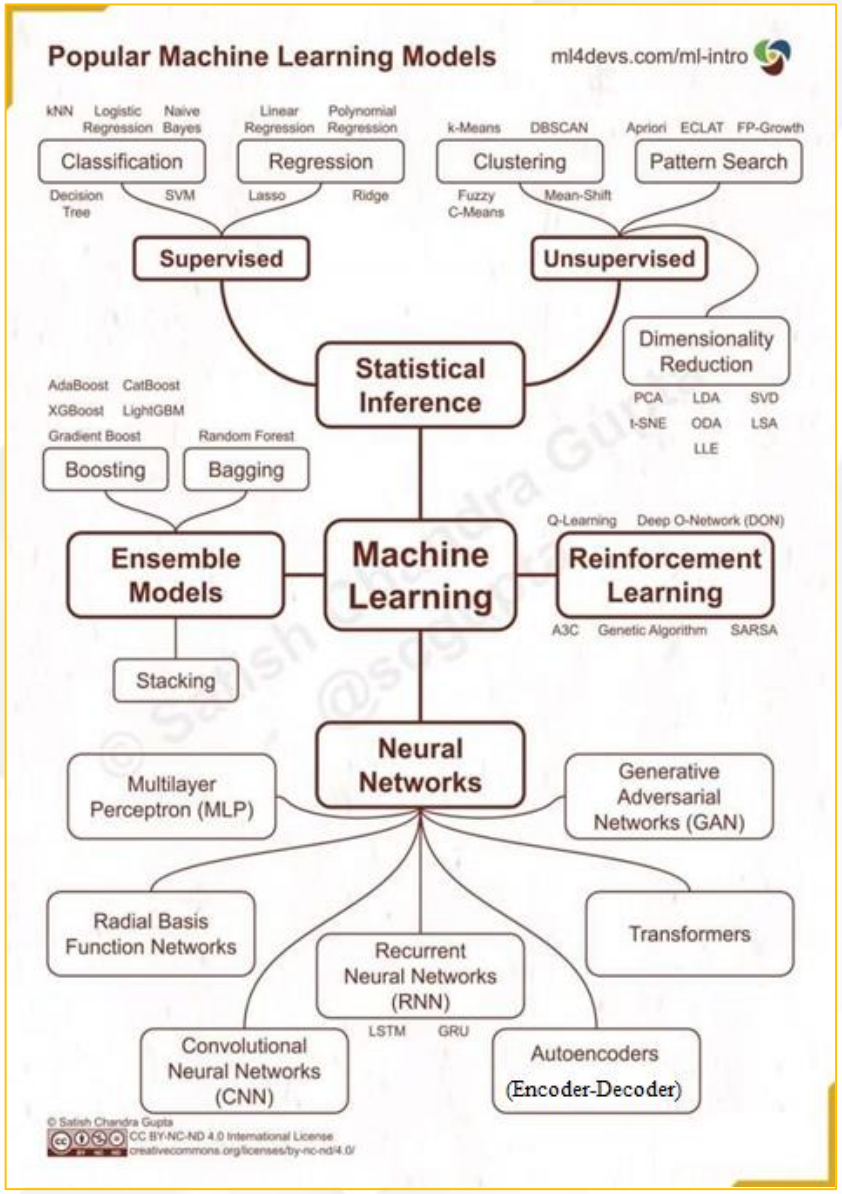
Asociación: Identificación de productos que habitualmente se compran juntos (análisis de la canasta). Ej. pan, azúcar -> leche. Reglas antecedente-consecuente. Algoritmos: A priori, FP Growth.

Todos los derechos reservados © Universidad Hemisferios

En la Figura 11 se muestra los tipos de modelos de machine learning existentes en la literatura.

En lo que sigue de la unidad se revisarán ciertos algoritmos predictivos de clasificación.

Figura 11. Tipos de modelos de machine learning



Fuente: ML4Devs (2020)

2.2 Configuración del Entorno de Python, Keras y TensorFlow

Siga el tutorial disponible aquí, para configurar localmente un entorno de desarrollo de aprendizaje automático utilizando la herramienta Anaconda Navigator. Anaconda es una distribución libre y abierta de los lenguajes Python y R, utilizada en ciencia de datos, y aprendizaje automático que permite instalar y administrar paquetes, dependencias y entornos para la Ciencias de Datos con Python de una manera muy sencilla.

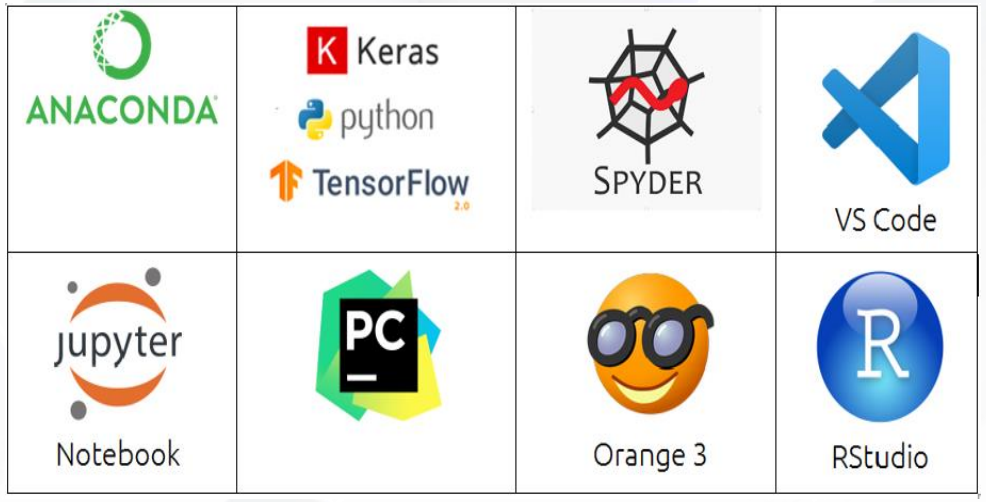
Todos los derechos reservados © Universidad Hemisferios

Ayuda a desarrollar proyectos de Ciencia de datos utilizando diversos entornos de desarrollo como Jupyter, PyCharm, Spyder, VS Code, Orange, RStudio, etc., como se illustran en la Figura 12.

En este tutorial se cubrirá los siguientes pasos:

- Descargar Anaconda navigator
- Instalar Anaconda
- Iniciar y actualizar Anaconda
- Actualizar la biblioteca scikit-learn
- Instalar bibliotecas de aprendizaje profundo

Figura 12. Entornos de desarrollo disponibles en Anaconda Navigator



Todos los derechos reservados © Universidad Hemisferios

Otra alternativa es el uso de un entorno de desarrollo gratuito en la nube como Google Colaboratory, o "Colab", que permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación.

Algunas de las principales características de Colab son:

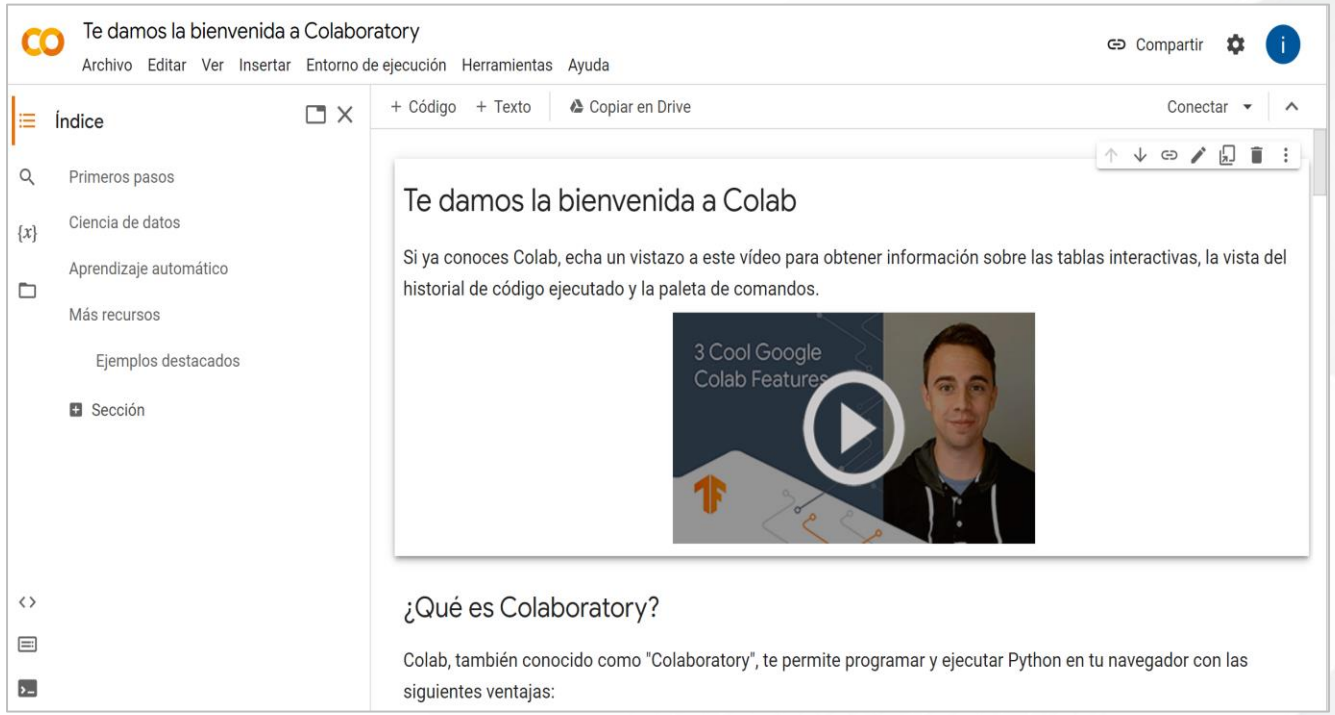
- Dado que se ejecuta en una máquina de Google, no es necesario realizar ninguna configuración.
- Google proporciona acceso gratuito a las GPU (limitadas).
- Es fácil de compartir, como cualquier archivo en el drive.

Todos los derechos reservados © Universidad Hemisferios

Lo primero que se debe hacer para utilizar el Google Colab es acceder a la siguiente dirección URL en donde se da la bienvenida a Colab e indica los primeros pasos y algunos recursos disponibles, como se indica la Figura 13.

https://colab.research.google.com/notebooks/intro.ipynb

Figura 13. Google Colab



En este punto siéntase libre de explorar el entorno de Colab teniendo una conexión a Internet.

Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

2.3 Conjuntos de Datos Disponibles

Existe una gran variedad de datasets públicamente disponibles para realizar tareas de análisis de datos. Algunos de ellos son los siguientes:

- UCI repo
- Kaggle
- Google dataset search
- sklearn.datasets
- https://www.datosabiertos.gob.ec/
- https://datos.gob.es/
- etc.

Todos los derechos reservados © Universidad Hemisferios

Tamaños de datasets:

Dataset pequeño: Conjunto de datos del orden de un millar (alrededor de mil) de ejemplos de entrenamiento.

Dataset mediano: Conjunto de datos del orden de decenas de miles de ejemplos de entrenamiento.

Dataset grande: Conjunto de datos del orden de centenas de miles o millones de ejemplos de entrenamiento.

Todos los derechos reservados © Universidad Hemisferios

2.4 Algoritmos de Clasificación

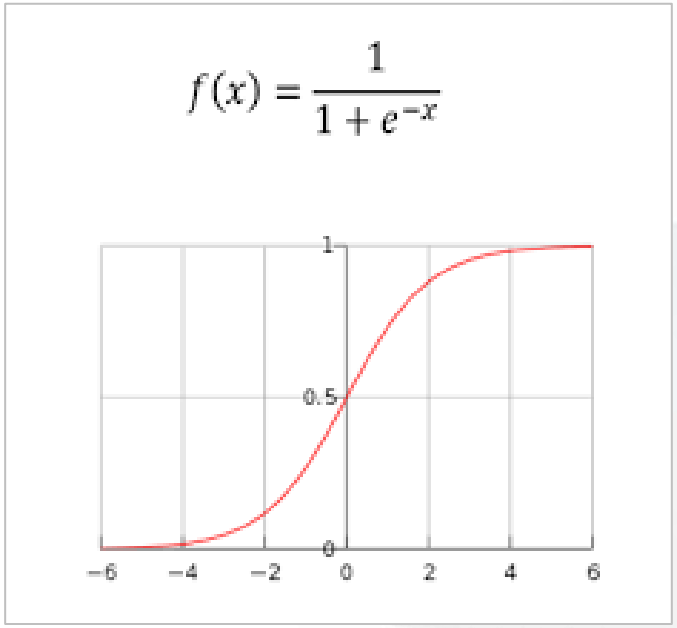
2.4.1 Regresión Logística

Es un método de aprendizaje automático supervisado utilizado para resolver problemas de clasificación binaria o multinomial (más de dos clases). Aunque su nombre incluye la palabra "regresión", en realidad se trata de un algoritmo de clasificación. Se utiliza comúnmente en una variedad de aplicaciones, como la detección de spam de correo electrónico, diagnósticos médicos, clasificación de documentos, entre otros.

Todos los derechos reservados © Universidad Hemisferios

Su nombre proviene de la función logística (también conocida como función sigmoide) que utiliza para modelar la relación entre las variables de entrada (características) y la probabilidad de que una instancia pertenezca a una clase específica, como se indica en la Figura 14.

Figura 14. Función logística (sigmoide)



La función logística tiene la siguiente forma:

$$P(Y=1) = \frac{1}{1+e^{-(a_0+a_1X_1+a_2X_2+\ldots+a_nX_n)}}$$

Donde:

- P(Y=1) es la probabilidad de pertenecer a la clase 1.
- e es la base del logaritmo natural.
- a0, a1, a2, ..., an son coeficientes que se aprenden (ajuste) durante el entrenamiento.
- X1, X2, ...,Xn son las variables de entrada.

Todos los derechos reservados © Universidad Hemisferios

Durante la fase de entrenamiento, el algoritmo de regresión logística ajusta los coeficientes a0, a1, a2, …,an de manera que el modelo se ajuste mejor a los datos de entrenamiento.

Una vez que el modelo está entrenado, se puede utilizar para hacer predicciones sobre nuevos datos, es decir, el modelo calcula la probabilidad de que esa instancia pertenezca a una de las clases. Si la probabilidad calculada es mayor que el umbral (generalmente 0.5), se clasifica como la clase positiva (1); de lo contrario, se clasifica como la clase negativa (0) en un problema de clasificación binaria.

A continuación, se presenta un ejemplo del algoritmo de regresión logística en Python, Keras y TensorFlow para la predicción del cáncer de mama (benigno=0, maligno=1) utilizando el conjunto de datos Breast Cancer Wisconsin (Diagnostic) disponible en la librería scikit-learn.

Todos los derechos reservados © Universidad Hemisferios

```
# Logistic regression for breast cancer
# importa librerías necesarias
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression

# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1

# Divide 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)

# Normaliza los datos para que tengan una escala similar
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```


Todos los derechos reservados © Universidad Hemisferios

```
# Crea y entrena el modelo de regresión logistica
model = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=100)
model.fit(X_train, y_train)

# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)

# Convierte las probabilidades en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))
```

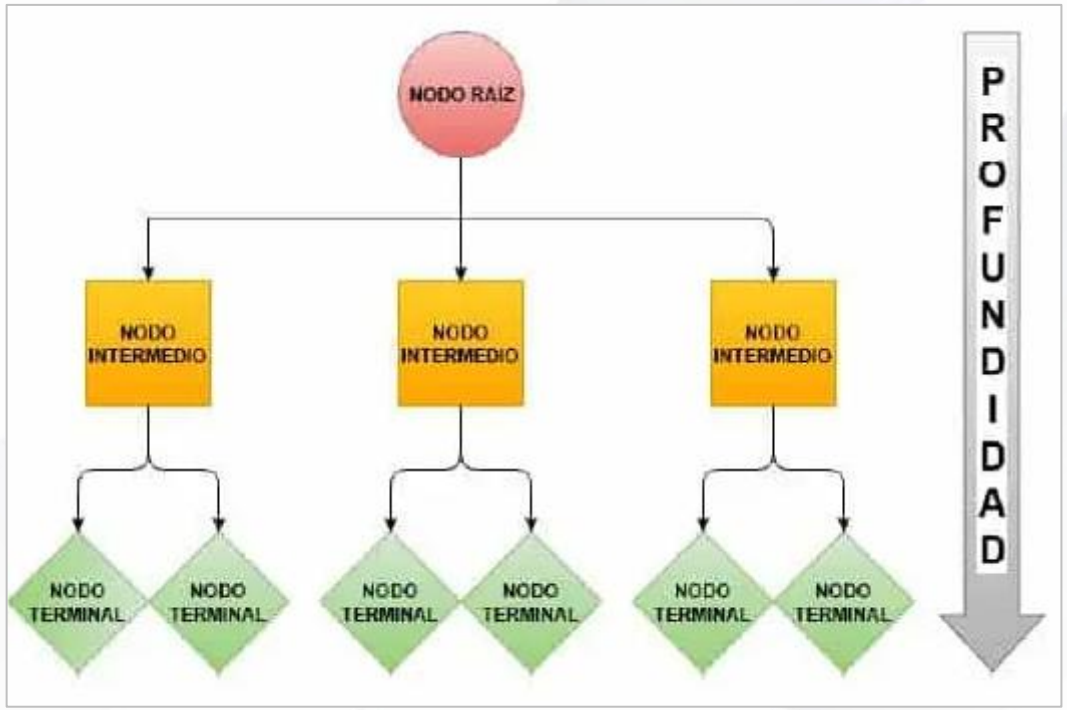
Tenga en cuenta que se pueden ajustar varios parámetros del algoritmo (como multi_class, solver, max_iter, etc.) para obtener un modelo óptimo. Revise la documentación (ayuda) del algoritmo para más detalles y experimente con ellos.

Todos los derechos reservados © Universidad Hemisferios

2.4.2 Árbol de decisión

Es un método de aprendizaje automático utilizado para la toma de decisiones (o hacer predicciones) y la resolución de problemas de clasificación y regresión. Este algoritmo se basa en la estructura de un árbol, donde cada nodo representa una decisión basada en una característica específica, y las ramas representan las posibles salidas o resultados de esa decisión como se indica en la Figura 15.

Figura 15. Estructura básica de un Árbol de decisión



Fuente: MaximaFormacion (2023)

Todos los derechos reservados © Universidad Hemisferios

Algunos conceptos clave relacionados con el algoritmo de árbol de decisión son los siguientes:

Nodo Raíz: El nodo superior del árbol, que representa la característica que mejor divide el conjunto de datos inicial en función de ciertos criterios (como la ganancia de información o la impureza).

Nodos Internos (intermedio): Los nodos en el medio del árbol que representan decisiones basadas en características específicas.

Hojas (nodo terminal): Los nodos finales del árbol que representan las etiquetas de clasificación o los valores de regresión resultantes.

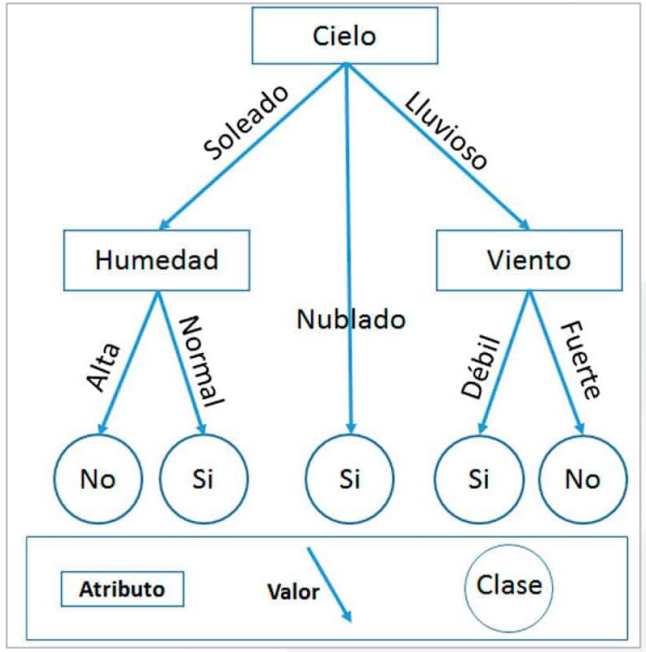
División: El proceso de dividir un nodo en dos o más nodos hijos según una característica específica. La elección de la característica y el criterio de división es un aspecto importante del algoritmo.

Criterios de Decisión: Los criterios utilizados para decidir cómo dividir los nodos, como la ganancia de información (entropía), la impureza Gini o el error cuadrado medio, dependiendo del tipo de problema (clasificación o regresión).

Todos los derechos reservados © Universidad Hemisferios

Un ejemplo de árbol de decisión para predecir si jugar o no un partido de tenis considerando variables atmosféricas se presenta en la Figura 16.

Figura 16. Árbol de decisión para predecir si jugar o no un partido de tenis



Fuente: InstitutoEmprende (2023)

Los árboles de decisión son populares en el aprendizaje automático debido a su simplicidad y capacidad para manejar conjuntos de datos tanto categóricos como numéricos. Además, son fácilmente interpretables y explicables, lo que los hace útiles en aplicaciones donde la transparencia del modelo es importante.

Todos los derechos reservados © Universidad Hemisferios

A continuación, se presenta un ejemplo del algoritmo de árbol de decisión en Python, Keras y TensorFlow para la predicción del cáncer de mama (benigno=0, maligno=1) utilizando el conjunto de datos Breast Cancer Wisconsin (Diagnostic) disponible en la librería scikit-learn.

```
# Decision Tree for breast cancer

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier

# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1

# Divide 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)
```


Todos los derechos reservados © Universidad Hemisferios

```
# Normaliza los datos para que tengan una escala
similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Crea y entrena el modelo de árbol de decisión
model = DecisionTreeClassifier(max_depth=4, criterion
= 'gini')
model.fit(X_train, y_train)

# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)

# Convierte las probabilidades en etiquetas binarias
(0 o 1)
y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo
entrenado
print(classification_report(y_test, y_pred))
```

Tenga en cuenta que se pueden ajustar varios parámetros del algoritmo (como max_depth, criterion, etc.) para obtener un modelo óptimo. Revise la documentación (ayuda) del algoritmo para más detalles y experimente con ellos.

Todos los derechos reservados © Universidad Hemisferios

2.4.3 Vecinos más cercanos (K-NN)

El algoritmo k-Nearest Neighbors (k-NN) es un método de aprendizaje automático supervisado utilizado para problemas de clasificación y regresión. La idea fundamental detrás del algoritmo k-NN es que los puntos de datos similares tienden a estar cerca unos de otros en el espacio de características.

Por lo tanto, para predecir una nueva instancia, el algoritmo busca los "k" puntos de datos más cercanos en el conjunto de entrenamiento y toma una decisión basada en la mayoría de las etiquetas de clase (en el caso de clasificación) o en el promedio (en el caso de regresión) de las etiquetas de los vecinos más cercanos.

Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

Algunos conceptos clave relacionados con el algoritmo k-NN son los siguientes:

Parámetro k:

El valor de "k" en k-NN representa el número de vecinos más cercanos que se deben considerar al hacer una predicción. Un valor pequeño de "k" (como 1) puede llevar a predicciones ruidosas y sensibles a valores atípicos, mientras que un valor grande de "k" suaviza las predicciones, pero puede perder detalles finos en los datos. La elección del valor de "k" es crítica en k-NN y puede afectar significativamente el rendimiento del modelo.

Función de Distancia:

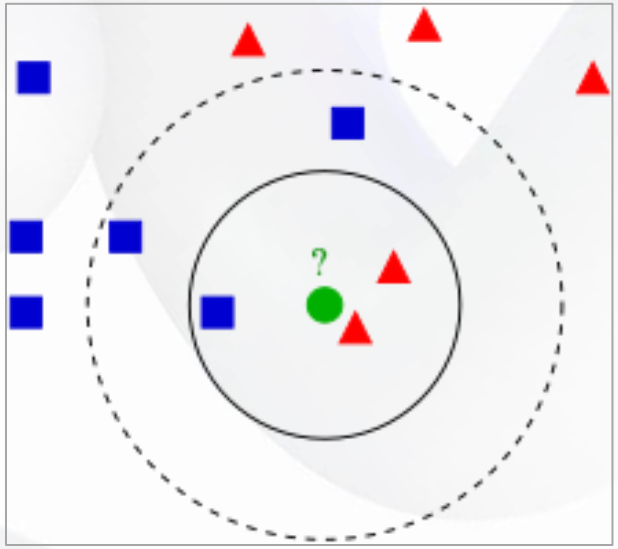
Para determinar qué puntos de datos son los más cercanos, se utiliza una función de distancia, como la distancia euclidiana, la distancia de Manhattan o la distancia de Minkowski. La elección de la función de distancia depende del problema y los datos.

Todos los derechos reservados © Universidad Hemisferios

Normalización:

Antes de aplicar k-NN, es importante normalizar las características para que todas tengan la misma influencia en la distancia entre puntos. Esto es especialmente importante cuando las características tienen escalas muy diferentes.

El algoritmo k-NN es simple de entender y de implementar, y es útil en una variedad de aplicaciones, como clasificación de documentos, recomendación de productos, diagnóstico médico y más. Sin embargo, es computacionalmente costoso en conjuntos de datos grandes, ya que requiere calcular la distancia entre la nueva instancia y todos los puntos de datos de entrenamiento.



Todos los derechos reservados © Universidad Hemisferios

A continuación, se presenta un ejemplo del algoritmo K-NN en Python, Keras y TensorFlow para la predicción del cáncer de mama (benigno=0, maligno=1) utilizando el conjunto de datos Breast Cancer Wisconsin (Diagnostic) disponible en la librería scikit-learn.

```
# K-NN for breast cancer
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1
```


Todos los derechos reservados © Universidad Hemisferios

```
# Divide 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)
# Normaliza los datos para que todas las
características tengan una escala similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Crea y entrena el modelo K-NN
model = KNeighborsClassifier(n_neighbors=3, p=2, #
Función euclidean
                             weights='distance')
model.fit(X_train, y_train)
# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)
# Convierte las probabilidades en etiquetas binarias
(0 o 1)
y_pred = (y_pred > 0.5)
# Muestra el informe de evaluación del modelo
entrenado print(classification_report(y_test, y_pred))
```

Tenga en cuenta que se pueden ajustar varios parámetros del algoritmo (como *n_neighbors, p, weights*, etc.) para obtener un modelo óptimo. Revise la documentación (ayuda) del algoritmo para más detalles y experimente con ellos.

Todos los derechos reservados © Universidad Hemisferios

2.4.4 Red Neuronal Artificial

Una Red Neuronal Artificial (RNA), o artificial neural network (ANN), es un modelo de aprendizaje automático inspirado en el funcionamiento del cerebro humano. Está compuesta por unidades de procesamiento llamadas neuronas artificiales o nodos, que están organizadas en capas interconectadas.

Las redes neuronales artificiales son utilizadas para abordar una amplia variedad de tareas de aprendizaje automático, incluyendo clasificación, regresión, procesamiento de imágenes, procesamiento de lenguaje natural y más.

Todos los derechos reservados © Universidad Hemisferios

POR FAVOR, EN ESTA PRIMERA SECCIÓN, ACTIVA TU AUDIO O USA TUS AUDÍFONOS

Todos los derechos reservados © Universidad Hemisferios

Red Neuronal Artificial

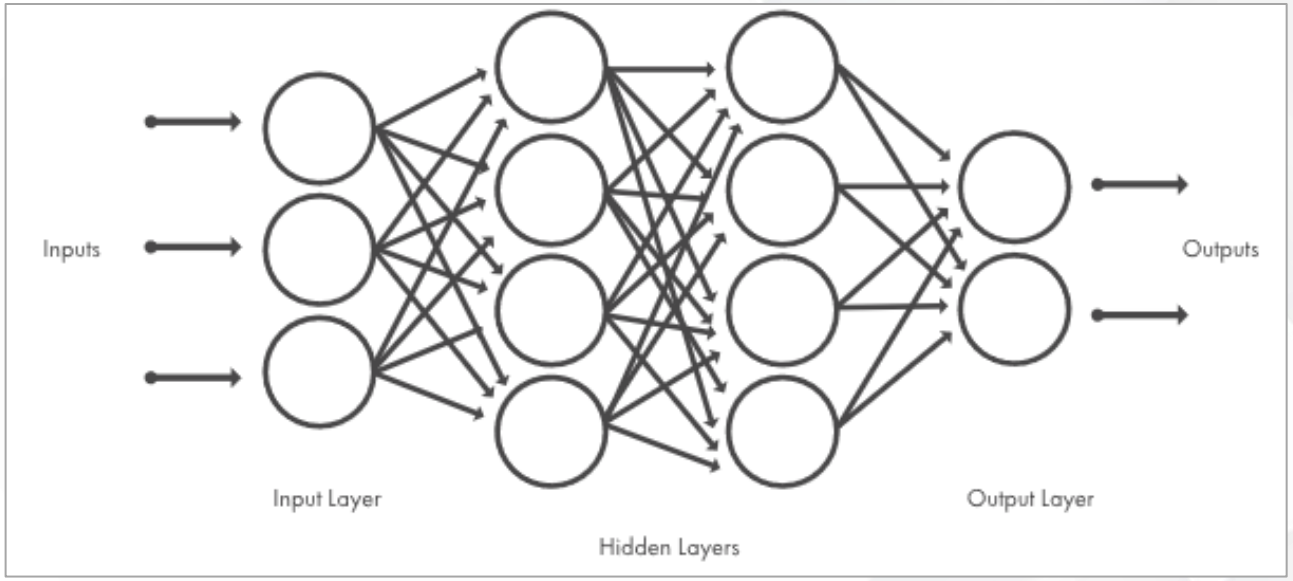
Haz clic para ver el video

Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

Arquitectura: Las redes neuronales pueden tener diferentes arquitecturas, como redes neuronales feed-forward (Figura 17), redes neuronales recurrentes (RNN) y redes neuronales convolucionales (CNN), cada una adaptada a tareas específicas. En este curso veremos las primeras, conocida como MLP (multi-layer perceptron).

Figura 17. Arquitectura típica de una red neuronal artificial



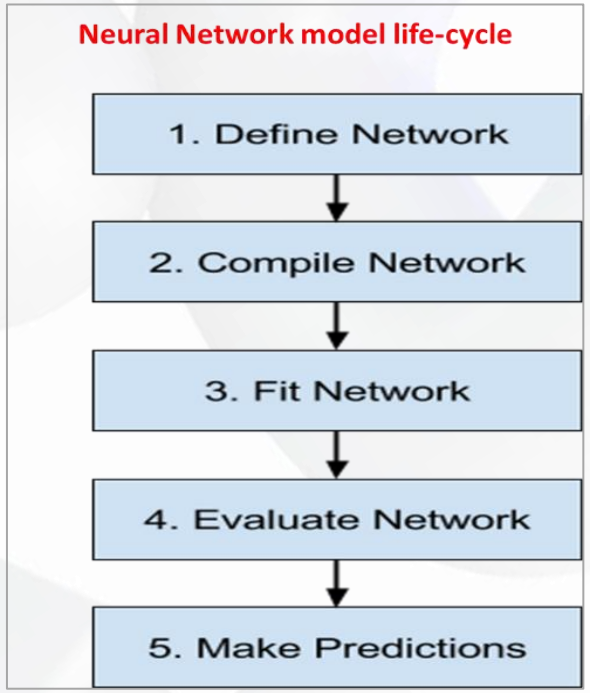
Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

Las redes neuronales artificiales se han destacado en una amplia gama de aplicaciones, incluyendo el procesamiento de imágenes, el procesamiento de lenguaje natural, la visión por computadora, la traducción automática, el reconocimiento de voz, la recomendación de contenido y muchas otras.

Su capacidad para aprender representaciones complejas y realizar tareas sofisticadas las ha convertido en un componente esencial del campo de la inteligencia artificial y el aprendizaje automático. En general, el ciclo de vida de un modelo de red neuronal artificial se muestra en la Figura 18.

Figura 18. Ciclo de vida de un modelo de red neuronal artificial



Todos los derechos reservados © Universidad Hemisferios

A continuación, se presenta un ejemplo del algoritmo RNA en Python, Keras y TensorFlow para la predicción del cáncer de mama (benigno=0, maligno=1) utilizando el conjunto de datos Breast Cancer Wisconsin (Diagnostic) disponible en la librería scikit-learn.

```
# RNA for breast cancer
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from keras.models import Sequential
from keras.layers.core import Dense
# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1
# Divide 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)
# Normaliza los datos para que tengan una escala similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```


Todos los derechos reservados © Universidad Hemisferios

```
# Crea y entrena el modelo RNA
model = Sequential()
model.add(Dense(10, activation='relu', input_dim=30))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=100)
# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)
# Convierte las probabilidades en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)
# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))
```

Tenga en cuenta que se pueden ajustar varios parámetros del algoritmo (como *activation, loss, optimizer, metrics*, epochs, etc.) para obtener un modelo óptimo. Revise la documentación (ayuda) del algoritmo para más detalles y experimente con ellos.

Todos los derechos reservados © Universidad Hemisferios

2.5 Evaluación de Modelos Predictivos y Métricas de Rendimiento

Estimar la bondad de un clasificador sirve para medir su capacidad de predicción sobre nuevas instancias (generalización). La validación se realiza brevemente basándose en la tasa de error, entendido como la clasificación incorrecta.

Tasa de error = # de errores cometidos / # total de casos

Todos los derechos reservados © Universidad Hemisferios

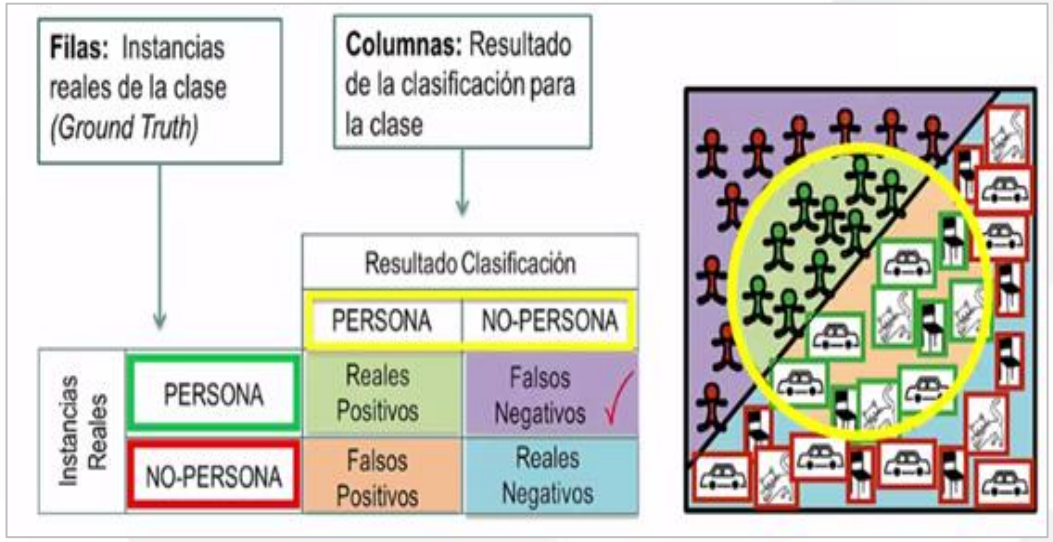
Una matriz de confusión permite ver, mediante una tabla de contingencias, la distribución de los errores (tipo I o falsos positivos y tipo II o falsos negativos) cometidos por un clasificador a lo largo de las distintas categorías del problema, ver Figura 18.

De una matriz de confusión se pueden extraer otros conceptos enriquecedores a la hora de comprender la distribución y naturaleza de los errores cometidos por el clasificador, así como algunas otras métricas de evaluación como: *exactitud (accuracy), precisión (precision), sensibilidad (recall), especificidad (specificity), F1-score, Curva ROC, área bajo la curva (AUC), coef. Kappa, coef. de determinación* R^2.

Todos los derechos reservados © Universidad Hemisferios

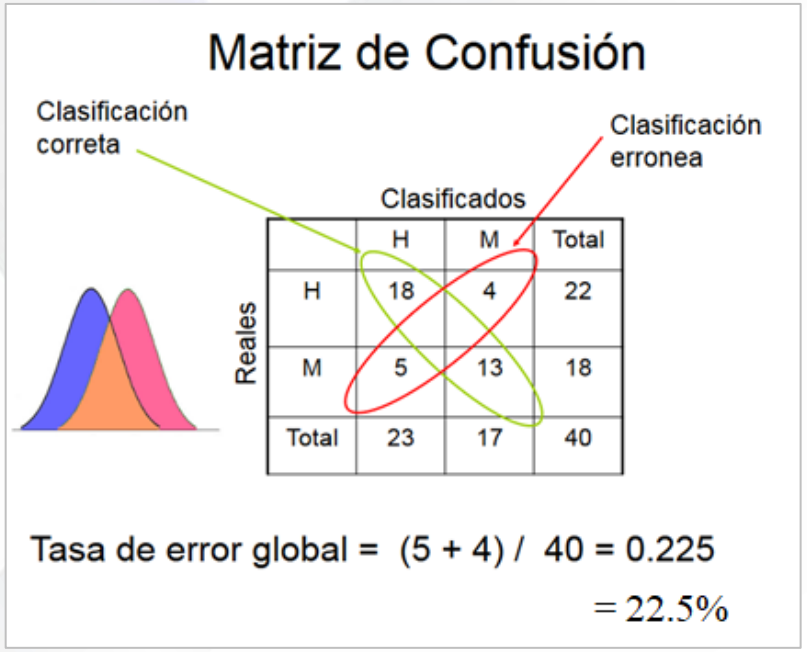
En la Figura 19 se muestra un ejemplo de un clasificador binario y la tasa de error cometido.

Figura 18. Matriz de confusión de un clasificador binario



Fuente: UAB (2016)

Figura 19. Ejemplo de matriz de confusión binaria



Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

Entendiendo los términos TP, FP, FN, TN:

- Verdadero positivo (TP): el modelo predijo positivo y es cierto (asignaciones positivas correctas)
- Verdadero Negativo (TN): El modelo predijo negativo y es cierto (asignaciones negativas correctas)
- Falso positivo (FP, error tipo 1): el modelo predijo positivo y es falso (asignaciones positivas incorrectas)
- Falso negativo (FN, error tipo 2): el modelo predijo negativo y es falso (asignaciones negativas incorrectas).

Todos los derechos reservados © Universidad Hemisferios

En un problema de clasificación binario, se denomina como la clase positiva (P ó 1) a los ejemplos que deseamos ser capaces de identificar. Los demás ejemplos los asignaremos a la clase negativa (N ó 0). Ejemplos:

- Drowsiness detection: drowsy (1), non-drowsy (0
- Breast cancer detection: malignant (1), benign (0)
- Fraudulent transaction detector: fraud (1), non-fraud (0)
- Spam filter: spam (1), non-spam (0)

Todos los derechos reservados © Universidad Hemisferios

Métricas de evaluación a partir de la matriz de confusión 2×2:

Exactitud (*accuracy*): calcula la tasa de aciertos totales (predicciones correctas) y da como resultado la eficacia del algoritmo.

Precisión (*precision*): calcula la relación entre las predicciones positivas correctas y los pronósticos positivos totales. Indica la calidad de la respuesta del clasificador.

Sensibilidad (o *Recall o TPR -True Positive Rate-*): calcula la tasa de positivos verdaderos (TP) e indica la eficiencia en la clasificación de todos los elementos que son de la clase positiva (+).

Especificidad: calcula la tasa de negativos verdaderos (TN) e indica la eficiencia en la clasificación de todos los elementos que son de la clase negativa (-).

Medida F1: conocida como media armónica de precisión y sensibilidad. Calcula el equilibrio entre la precisión y la sensibilidad, que varía de 0 a 1, siendo 1 el mejor valor.

Todos los derechos reservados © Universidad Hemisferios

En la Tabla 5 se presenta la matriz de confusión para la detección de somnolencia. Fíjese que la clase negativa está en la primera columna, a diferencia de la mostrada en la Figura 18. Ambas formas de organización son usadas y se deben considerar en el momento de calcular las métricas de evaluación de un algoritmo.

Tabla 5. Estructura de la Matriz de confusión para la detección de somnolencia.

		PREDICHO	
		No-Somnolencia	Somnolencia
REAL	No-Somnolencia	TN	FP
	Somnolencia	FN	TP

Todos los derechos reservados © Universidad Hemisferios

Las fórmulas para calcular las métricas de evaluación (a partir de la Tabla 5) son las siguientes:

$$accuracy = \frac{TP + TN}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP} + \mathrm{TN}}$$

$$precision = \frac{TP}{\mathrm{TP} + \mathrm{FP}}$$

$$sensitivity\ (o\ recall) = \frac{TP}{\mathrm{TP} + \mathrm{FN}}$$

$$specificity = \frac{TN}{\mathrm{TN} + \mathrm{FP}}$$

$$F1\ score = \frac{2 * precisión * recall}{precisión + recall}$$

Todos los derechos reservados © Universidad Hemisferios

En la Tabla 6 se muestra un ejemplo de un clasificador binario para el cálculo de las métricas de evaluación.

Tabla 6. Matriz de confusión para la detección de somnolencia.

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

¿Qué se puede aprender de esta matriz de confusión?

- De esos 165 casos, el clasificador predijo "sí" 110 veces y "no" 55 veces.
- En realidad, 105 pacientes de la muestra padecen la enfermedad y 60 pacientes no padecen.

Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

Métricas:

- Accuracy: (TP+TN)/total = (100+50)/165 = 0.91
- Error Rate: (FP+FN)/total = (10+5)/165 = 0.09
- Precision: TP/predicted yes = 100/110 = 0.91
- Sensitivity" or "Recall“ (True Positive Rate): TP/actual yes = 100/105 = 0.95
- Specificity (True Negative Rate:): TN/actual no = 50/60 = 0.83
- F1-score = (2*0.91*0.95)/(0.91+0.95)= 0.93

Todos los derechos reservados © Universidad Hemisferios

A continuación, se presenta a modo de ejemplo la evaluación de modelos en Python, la librería Scikit-learn y los resultados obtenidos (Figura 20) en un dataset pequeño conteniendo 6 observaciones:

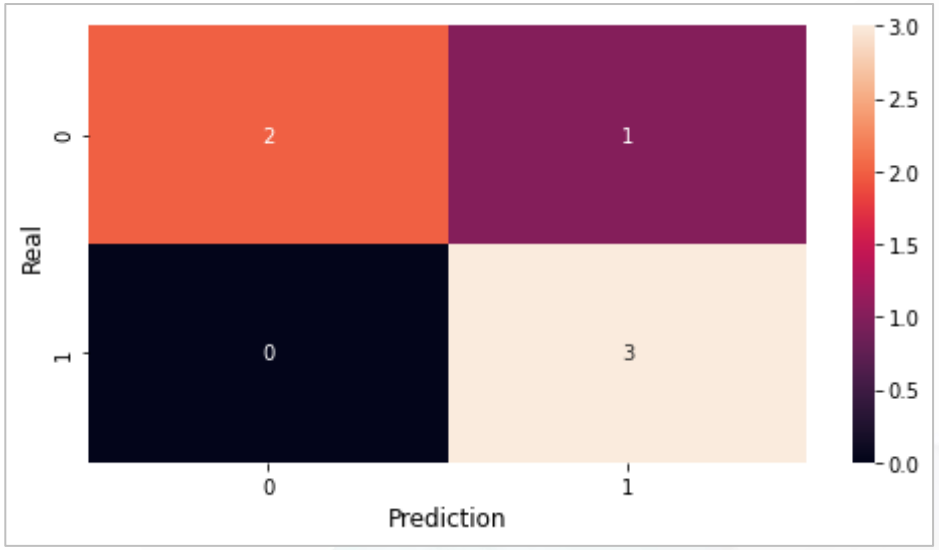
```
# Dataset:
y_true = [0, 1, 0, 0, 1, 1]
y_pred = [0, 0, 1, 0, 0, 1]
# Matriz de confusión:
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
# Exactitud:
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
# Sensibilidad:
from sklearn.metrics import recall_score
recall_score(y_true, y_pred)
# Especificidad:
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, pos_label=0)
# Precisión:
from sklearn.metrics import precision_score
precision_score(y_true, y_pred)
```

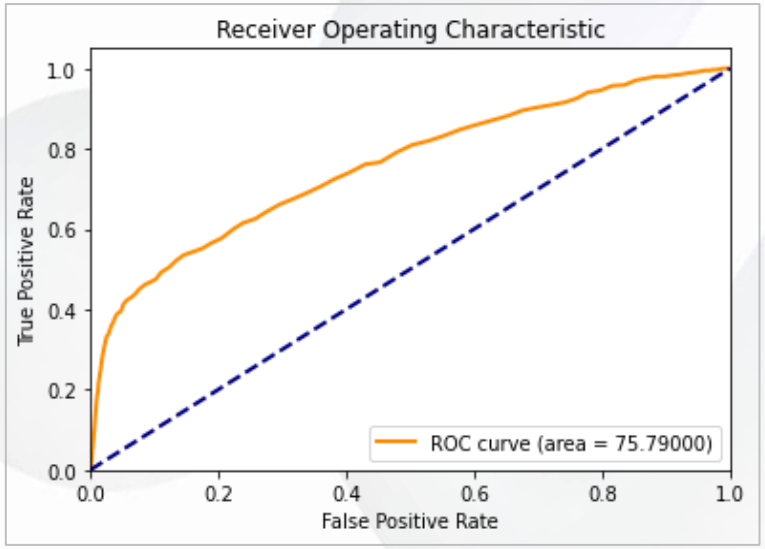

Todos los derechos reservados © Universidad Hemisferios

```
# Puntuación F1:
from sklearn.metrics import f1_score
f1_score(y_true, y_pred)
# Área bajo la curva:
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_true, y_pred)
# R Score:(R^2 coefficient of determination)
from sklearn.metrics import r2_score
r2_score(y_true, y_pred)
# ROC Curve (binary classification problem):
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
plt.plot(roc_curve(y_true, y_pred)[0], roc_curve(y_true, y_pred)[1],
color='darkorange',lw=2, label='ROC curve (area = %0.2f)' % auc)
```


Todos los derechos reservados © Universidad Hemisferios

Figura 20. Resultados del Código Python mostrando la matriz de confusión y la curva ROC.





La matriz de confusión y la curva ROC se puede adaptar para problemas de clasificación con más de 2 clases. Sin embargo, esto queda como trabajo autónomo del maestrante.

Escuela de Educación Online

Todos los derechos reservados © Universidad Hemisferios

uhemisferios uhe.oficial uhe_oficial Universidad Hemisferios

uhemisferios.edu.ec