Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución

TC3006C. Inteligencia artificial avanzada para la ciencia de datos I

Grupo: 101

A01749667. Alan Contreras Prieto

Prof. Jorge Adolfo Ramírez Uresti

Fecha de entrega: 10 de septiembre de 2025

Semestre agosto – diciembre 2025

Etapa con Dataset anterior:

Dataset:

Para entrenar al modelo se utilizó el mismo procedimiento que la entrega anterior. Un dataset sintético generado con la función make_classification() de sklearn. Los parámetros seleccionados fueron:

- n_samples = 10000 para contar con un conjunto de datos amplio.
- n_features = 5 de manera que la red tuviera múltiples entradas y se pudieran experimentar diferentes arquitecturas.
- n_classes = 2 para realizar una clasificación binaria.

La partición de datos se realizó con la función train_test_split() de sklearn, dividiendo el conjunto en 80% para entrenamiento y 20% para prueba.

Modelo: Árbol de decisión

Se implementó un árbol de decisión utilizando la clase DecisionTreeClassifier de sklearn. Para encontrar la configuración más adecuada se realizó un GridSearchCV, evaluando las siguientes combinaciones de hiperparámetros:

- criterion = {gini, entropy, log_loss}: este parámetro define la función de impureza o de ganancia de información utilizada para dividir los nodos.
    - Gini: mide la probabilidad de clasificar incorrectamente un punto.
    - Entropy: utiliza la teoría de la información para medir la pureza.
    - Log_loss: está más orientado a probabilidades bien calibradas en las predicciones.
- splitter = {best, random}: determina la estrategia para elegir la mejor división.
    - Best: selecciona el mejor split posible en cada nodo.
    - Random: selecciona divisiones de forma aleatoria, lo que puede introducir más variabilidad en los árboles.
- max_depth = {5, 10, 15, 20, 25, 30}: establece la profundidad máxima del árbol. Limitar la profundidad es importante para evitar el sobreajuste, pues árboles muy profundos pueden memorizar el dataset en lugar de generalizar.

- min_samples_split = {50, 100, 250, 500}: controla el número mínimo de muestras requeridas para dividir un nodo. Valores más altos producen árboles más simples y reducen el riesgo de sobreajuste.

Evaluación con matriz de confusión y métricas:

Para evaluar el desempeño de cada arquitectura se consideraron tanto la matriz de confusión como las principales métricas de clasificación:

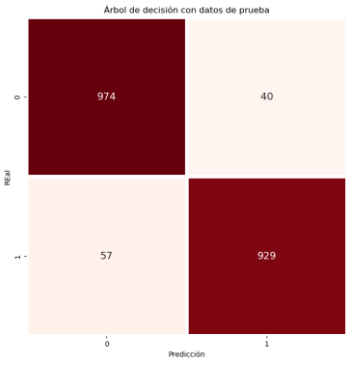
1. Accuracy (exactitud): proporción de aciertos sobre el total de predicciones.
2. Precision (precisión): proporción de verdaderos positivos sobre las predicciones positivas.
3. Recall (sensibilidad): proporción de verdaderos positivos sobre los positivos reales.
4. F1-score: promedio armónico entre precision y recall.

Durante la experimentación, los mejores hiperparámetros no fueron siempre los mismos en cada ejecución. Esto se debe a que tanto la generación del dataset (make_classification) como la partición en conjuntos de entrenamiento y prueba (train_test_split) tienen un componente aleatorio. Por lo tanto, diferentes particiones pueden favorecer a distintas configuraciones de hiperparámetros. Sin embargo, se observó que los modelos con profundidades intermedias (entre 5 y 15) y valores de min_samples_split relativamente altos (100–500) tienden a obtener un desempeño más estable, con accuracies que rondan entre 0.92 y 0.95. Esto confirma que la red neuronal de la práctica anterior y el árbol de decisión en esta práctica son capaces de resolver el problema de clasificación binaria de forma adecuada, aunque los resultados dependen de la aleatoriedad en la preparación del dataset.

Mejores resultados de la última corrida:

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.94	0.96	0.95	1014
1	0.96	0.94	0.95	986
accuracy			0.95	2000
macro avg	0.95	0.95	0.95	2000
weighted avg	0.95	0.95	0.95	2000



Etapa con nuevo Dataset:

Dado que la implementación con el mismo Dataset fue muy sencilla, he decidido aumentar la complejidad del problema generando un dataset más grande y multiclase.

Dataset:

Para entrenar al modelo se utilizó un dataset sintético generado con la función make_classification() de sklearn. Los parámetros seleccionados fueron:

- n_samples = 50000: se generó un conjunto de datos más amplio con el fin de disponer de aproximadamente 10,000 ejemplos por clase. Esto permite un mejor entrenamiento de modelos más complejos como Random Forest y reduce la varianza de los resultados.
- n_features = 10: se eligieron diez características para incrementar la dimensionalidad del problema y hacer más desafiante la clasificación. Con más atributos, el modelo tiene que aprender relaciones más complejas entre las variables.
- n_classes = 5: a diferencia de la práctica anterior (binaria), aquí se planteó un problema de clasificación multiclase, lo cual requiere que el modelo diferencie entre cinco categorías distintas.
- n_informative = 5: se estableció que solo la mitad de las características aporten información relevante para la clasificación. Esto obliga al modelo a identificar qué atributos son realmente útiles y cuáles son ruido, simulando un escenario más realista.

La partición de datos se realizó con la función train_test_split() de sklearn, dividiendo el conjunto en 80% para entrenamiento y 20% para prueba.

Modelo: Random forest

Para abordar este problema se utilizó un Random Forest Classifier, un ensamble de múltiples árboles de decisión que mejora la precisión al promediar los resultados de varios modelos.

Se realizó una búsqueda de hiperparámetros con GridSearchCV, considerando las siguientes combinaciones:

- n_estimators = {10, 50, 100}: este parámetro indica el número de árboles en el bosque. Inicialmente se probaron valores más altos, pero la ejecución de GridSearchCV resultaba demasiado lenta, por lo que se redujo la búsqueda a tres valores representativos.
- criterion = {gini, entropy, log_loss}: define la métrica utilizada para medir la calidad de las divisiones en los nodos.
- max_depth = {10, 20, 30, None}: limita la profundidad máxima de los árboles. Usar None significa que los nodos se expanden hasta que todas las hojas sean puras o tengan menos muestras que el mínimo requerido.
- min_samples_split = {50, 100, 250, 500}: determina el número mínimo de ejemplos requeridos para dividir un nodo. Valores más altos generan árboles más simples y con menor riesgo de sobreajuste.

Evaluación con matriz de confusión y métricas:

Para evaluar el desempeño de cada arquitectura se consideraron tanto la matriz de confusión como las principales métricas de clasificación:

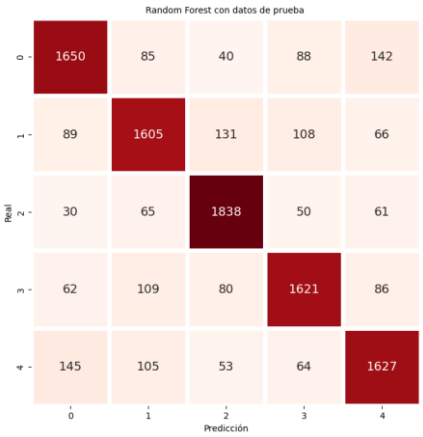
1. Accuracy (exactitud): proporción de aciertos sobre el total de predicciones.
2. Precision (precisión): proporción de verdaderos positivos sobre las predicciones positivas.
3. Recall (sensibilidad): proporción de verdaderos positivos sobre los positivos reales.
4. F1-score: promedio armónico entre precision y recall.

El Random Forest final alcanzó un accuracy de 0.83, lo cual indica que el modelo fue capaz de clasificar correctamente la mayoría de los ejemplos. Aunque no es perfecto, se trata de

un desempeño decente considerando la complejidad añadida por el mayor número de clases y la presencia de variables no informativas.

Reporte de Clasificación:

	precision	recall	f1-score	support
Clase 0	0.84	0.82	0.83	2005
Clase 1	0.82	0.80	0.81	1999
Clase 2	0.86	0.90	0.88	2044
Clase 3	0.84	0.83	0.83	1958
Clase 4	0.82	0.82	0.82	1994
accuracy			0.83	10000
macro avg	0.83	0.83	0.83	10000
weighted avg	0.83	0.83	0.83	10000



Conclusiones:

La implementación de Random Forest en un problema de clasificación multiclase demostró que este modelo es robusto y capaz de manejar datasets grandes y con ruido. Si bien el accuracy de 0.83 no es tan alto como en el caso binario de la primera etapa, es un resultado esperado al incrementar la dificultad del problema.