Tecnológico de Monterrey

Campus:
Monterrey

Inteligencia Artificial Avanzada para la Ciencia de Datos (Gpo 102)

Curso:
TC3006C.102

Módulo 2: Portafolio 2

Daniel Sánchez Villarreal A01197699

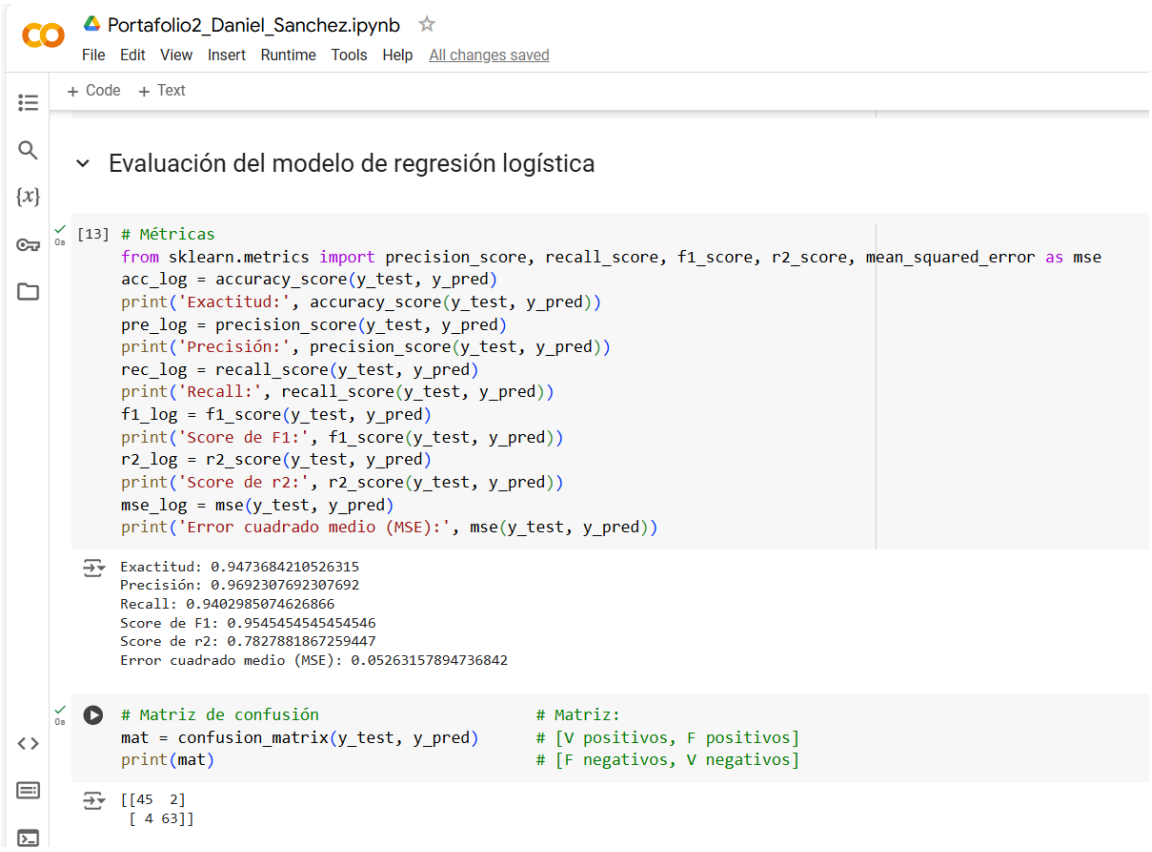
Lugar y Fecha:
Monterrey, Nuevo León
7 de septiembre de 2024

En este Portafolio 2, decidí usar el dataset de breast_cancer y entrenar 6 modelos, los cuales fueron regresión logística, árboles de decisión, random forest, KNN, SVC, y Gaussian Naive Bayes. Esto con el fin de evaluar cuál de todos tiene el mejor desempeño y ese sería el que escogería como mi mejor modelo para después usar GridSearch para buscar e implementarle los mejores hiperparámetros y evaluar su rendimiento para finalmente hacer algunas predicciones con ese modelo mejorado.

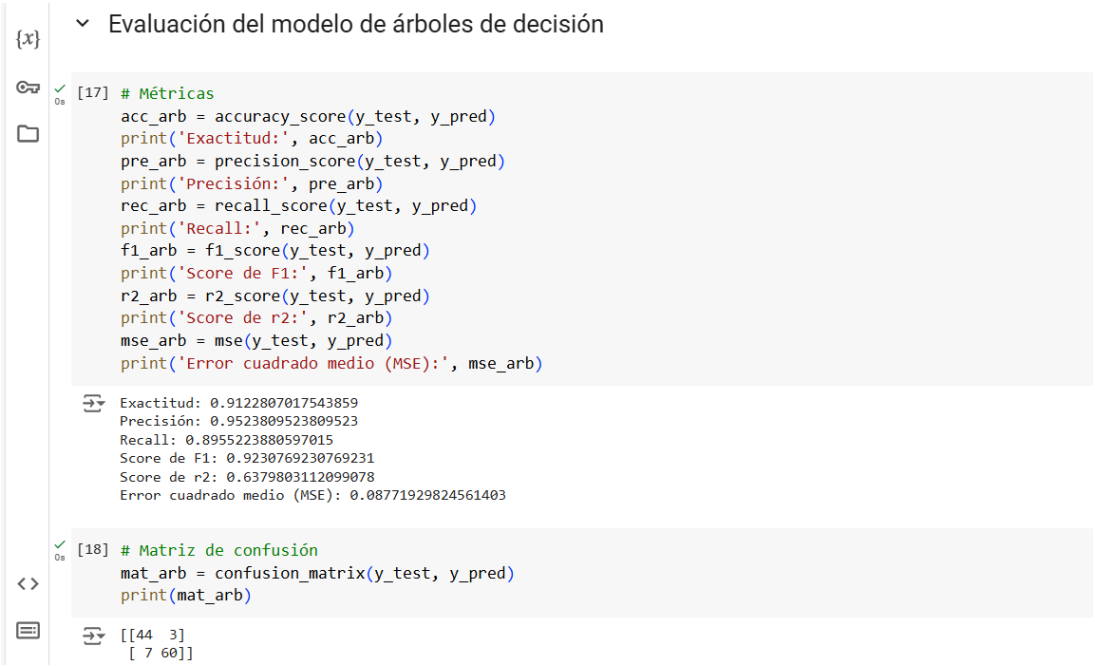
Para esto, lo primero que hice fue la preparación de los datos. En este caso, definí la Y como la variable target, y la X para el resto de features en el dataset. Usé target_names para ver la clase de datos para la variable target, los cuales con este comando pude observar que cada instancia (sample) tenía un valor ya sea de benigno o maligno en esa variable target. Entonces lo que hice a continuación fue transformar esos datos para establecer benigno como 0 y maligno como 1 en todas las instancias de dicha variable target en el dataset.

Posteriormente, verifiqué si había o no datos nulos en el dataset y tras ver que no había ningún dato nulo en ninguna de las features, procedí con el entrenamiento de mis 6 modelos para después evaluarlos todos para poder saber cuál de todos tiene el mejor rendimiento y ese será el que voy a elegir para usar GridSearch para implementarle los mejores hiperparámetros y evaluar su rendimiento para después hacer predicciones con dicho modelo.

Comencé con el modelo de regresión logística, el cual tras evaluar sus métricas obtuve los siguientes resultados:



Después, en el de árboles de decisión obtuve lo siguiente:



Luego, obtuve los siguientes resultados en el de random forest (este fue el mejor):

Evaluación del modelo de Random Forest

```
[21] # Métricas
     acc_rf = accuracy_score(y_test, y_pred)
     print('Exactitud:', acc_rf)
     pre_rf = precision_score(y_test, y_pred)
     print('Precisión:', pre_rf)
     rec_rf = recall_score(y_test, y_pred)
     print('Recall:', rec_rf)
     f1_rf = f1_score(y_test, y_pred)
     print('Score de F1:', f1_rf)
     r2_rf = r2_score(y_test, y_pred)
     print('Score de r2:', r2_rf)
     mse_rf = mse(y_test, y_pred)
     print('Error cuadrado medio (MSE):', mse_rf)
```

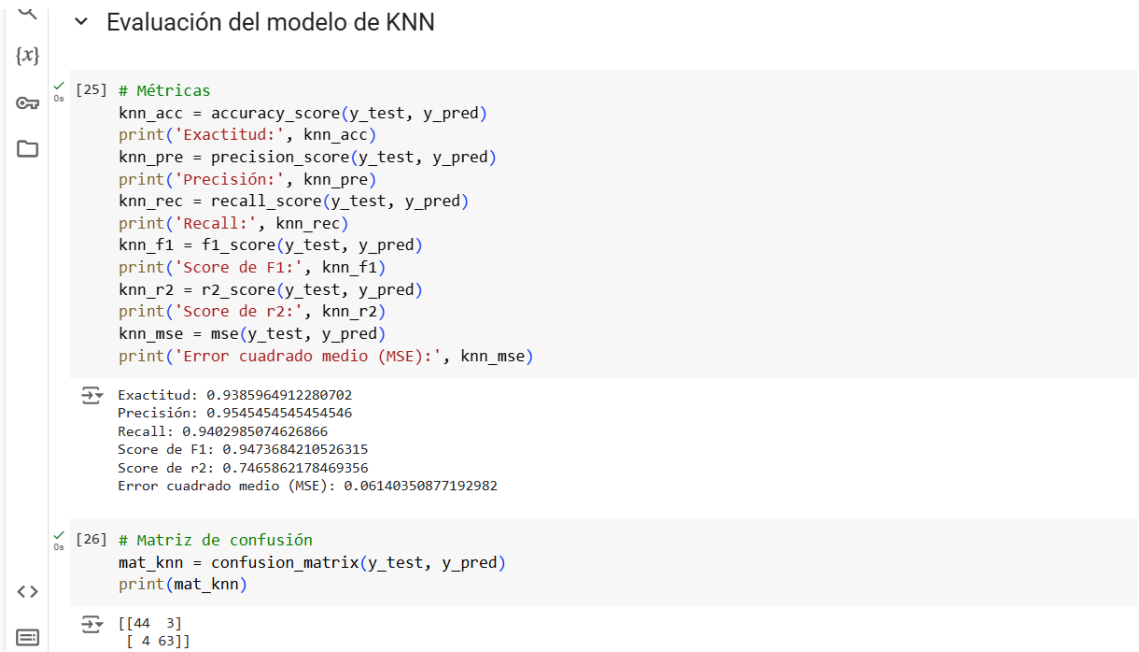
```
Exactitud: 0.956140350877193
Precisión: 0.9696969696969697
Recall: 0.9552238805970149
Score de F1: 0.9624060150375939
Score de r2: 0.8189901556049539
Error cuadrado medio (MSE): 0.043859649122807015
```

```
[22] # Matriz de confusión
     mat_rf = confusion_matrix(y_test, y_pred)
     print(mat_rf)
```

```
[[45  2]
 [ 3 64]]
```

✓ 26s completed at 11:30 PM

Después seguí con el modelo de K vecinos más cercanos (KNN):



Evaluación del modelo de KNN

```
[25] # Métricas
     knn_acc = accuracy_score(y_test, y_pred)
     print('Exactitud:', knn_acc)
     knn_pre = precision_score(y_test, y_pred)
     print('Precisión:', knn_pre)
     knn_rec = recall_score(y_test, y_pred)
     print('Recall:', knn_rec)
     knn_f1 = f1_score(y_test, y_pred)
     print('Score de F1:', knn_f1)
     knn_r2 = r2_score(y_test, y_pred)
     print('Score de r2:', knn_r2)
     knn_mse = mse(y_test, y_pred)
     print('Error cuadrado medio (MSE):', knn_mse)
```

```
Exactitud: 0.9385964912280702
Precisión: 0.9545454545454546
Recall: 0.9402985074626866
Score de F1: 0.9473684210526315
Score de r2: 0.7465862178469356
Error cuadrado medio (MSE): 0.06140350877192982
```

```
[26] # Matriz de confusión
     mat_knn = confusion_matrix(y_test, y_pred)
     print(mat_knn)
```

```
[[44  3]
 [ 4 63]]
```

Luego, en el modelo de SVC, obtuve las siguientes métricas:

Evaluación del modelo de SVC

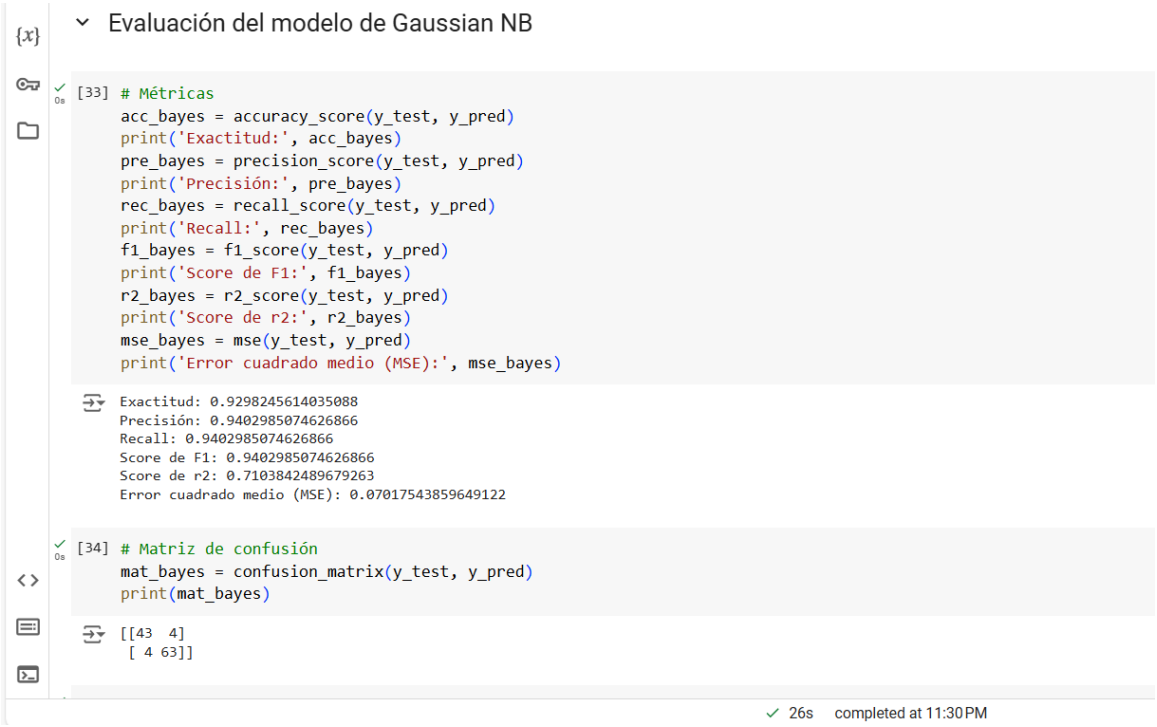
```
# Métricas
acc_svc = accuracy_score(y_test, y_pred)
print('Exactitud:', acc_svc)
pre_svc = precision_score(y_test, y_pred)
print('Precisión:', pre_svc)
rec_svc = recall_score(y_test, y_pred)
print('Recall:', rec_svc)
f1_svc = f1_score(y_test, y_pred)
print('Score de F1:', f1_svc)
r2_svc = r2_score(y_test, y_pred)
print('Score de r2:', r2_svc)
mse_svc = mse(y_test, y_pred)
print('Error cuadrado medio (MSE):', mse_svc)
```

```
Exactitud: 0.9298245614035088
Precisión: 0.9041095890410958
Recall: 0.9850746268656716
Score de F1: 0.9428571428571428
Score de r2: 0.7103842489679263
Error cuadrado medio (MSE): 0.07017543859649122
```

```
# Matriz de confusión
mat_svc = confusion_matrix(y_test, y_pred)
print(mat_svc)
```

```
[[40  7]
 [ 1 66]]
```

Finalmente, en el modelo de Gaussian Naive Bayes obtuve el siguiente rendimiento:



Evaluación del modelo de Gaussian NB

```
[33] # Métricas
acc_bayes = accuracy_score(y_test, y_pred)
print('Exactitud:', acc_bayes)
pre_bayes = precision_score(y_test, y_pred)
print('Precisión:', pre_bayes)
rec_bayes = recall_score(y_test, y_pred)
print('Recall:', rec_bayes)
f1_bayes = f1_score(y_test, y_pred)
print('Score de F1:', f1_bayes)
r2_bayes = r2_score(y_test, y_pred)
print('Score de r2:', r2_bayes)
mse_bayes = mse(y_test, y_pred)
print('Error cuadrado medio (MSE):', mse_bayes)
```

```
Exactitud: 0.9298245614035088
Precisión: 0.9402985074626866
Recall: 0.9402985074626866
Score de F1: 0.9402985074626866
Score de r2: 0.7103842489679263
Error cuadrado medio (MSE): 0.07017543859649122
```

```
[34] # Matriz de confusión
mat_bayes = confusion_matrix(y_test, y_pred)
print(mat_bayes)
```

```
[[43  4]
 [ 4 63]]
```

✓ 26s completed at 11:30 PM

Una vez entrenados los 6 modelos, continúo con la siguiente parte, que es tomar la decisión sobre cuál tuvo mejor rendimiento para en base a esto elegir el modelo final, en el cual usaré Grid Search para encontrar los mejores hiperparámetros de dicho modelo e implementarlo de esta forma para finalmente evaluar su rendimiento y hacer predicciones con este.

Esta importante decisión la tomé en base al siguiente análisis, en el cual hice una tabla donde se muestran todas las métricas evaluadas (accuracy, precision, recall, F1, r2, y MSE) para los 6 modelos que entrené, tal como se puede ver a continuación:

Modelo	Exactitud	Precisión	Recall	Score F1	Score r2	MSE
Regresión logística	0.9473684210526315	0.9692307692307692	0.9402985074626866	0.9545454545454546	0.7827881867259447	0.05263157894736
Árboles de decisión	0.9122807017543859	0.9523809523809523	0.8955223880597015	0.9230769230769231	0.6379803112099078	0.08771929824561
Random Forest	0.956140350877193	0.9696969696969697	0.9552238805970149	0.9624060150375939	0.8189901556049539	0.04385964912280
KNN	0.9385964912280702	0.9545454545454546	0.9402985074626866	0.9473684210526315	0.7465862178469356	0.06140350877192
SVC	0.9298245614035088	0.9041095890410958	0.9850746268656716	0.9428571428571428	0.7103842489679263	0.07017543859649
Gaussian Naive Bayes	0.9298245614035088	0.9402985074626866	0.9402985074626866	0.9402985074626866	0.7103842489679263	0.07017543859649

```
# Escogeré el modelo que tenga menos inexactitud (1 - todas las métricas) y el menor MSE.
## El modelo que tenga la menor cifra de inexactitud es el que ganará y le aplicaré grid search.
log_inexact = ( (1-acc_log) + (1-pre_log) + (1-rec_log) + (1-f1_log) + (1-r2_log) + mse_log)
arb_inexact = ( (1-acc_arb) + (1-pre_arb) + (1-rec_arb) + (1-f1_arb) + (1-r2_arb) + mse_arb)
rf_inexac = ( (1-acc_rf) + (1-pre_rf) + (1-rec_rf) + (1-f1_rf) + (1-r2_rf) + mse_rf)
knn_inexac = ( (1-knn_acc) + (1-knn_pre) + (1-knn_rec) + (1-knn_f1) + (1-knn_r2) + knn_mse)
svc_inexac = ( (1-acc_svc) + (1-pre_svc) + (1-rec_svc) + (1-f1_svc) + (1-r2_svc) + mse_svc)
bayes_inexac = ( (1-acc_bayes) + (1-pre_bayes) + (1-rec_bayes) + (1-f1_bayes) + (1-r2_bayes) + mse_bayes)
print("Modelo de regresión logística:", log_inexact)
print("Modelo de árboles de decisión:", arb_inexact)
print("Modelo de random forest:", rf_inexac)
print("Modelo de KNN:", knn_inexac)
print("Modelo de SVC:", svc_inexac)
print("Modelo de Gaussian Naive Bayes:", bayes_inexac)
lista_inexac = [log_inexact, arb_inexact, rf_inexac, knn_inexac, svc_inexac, bayes_inexac]
ganador = min(lista_inexac)
print("El modelo elegido es el que tiene menor índice de inexactitud (mayor exactitud en sus métricas en general). En este caso
```

```
Modelo de regresión logística: 0.4584002399298817
Modelo de árboles de decisión: 0.7664780217637432
Modelo de random forest: 0.3814022773090816
Modelo de KNN: 0.5340084166361513
Modelo de SVC: 0.5979252694611459
Modelo de Gaussian Naive Bayes: 0.6090711058369963
El modelo elegido es el que tiene menor índice de inexactitud (mayor exactitud en sus métricas en general). En este caso el elegido es: 0.38140227
```

Una vez realizado este análisis, se puede observar que el modelo que tuvo mejor rendimiento en todas sus métricas en general fue el modelo de Random Forest (esto se confirma también en la tabla de resumen de métricas donde se puede observar que el de Random Forest es el que tiene más precisión y menor MSE). Por lo tanto, como dije anteriormente, este fue el modelo que elegí para hacerle Grid Search e implementar los mejores hiperparámetros encontrados en este modelo para finalmente hacer algunas predicciones con este.

Para esto, utilicé el Grid Search configurado para probar el modelo, poniendo como hiperparámetros 100, 200 y 300 con respecto a la cantidad de árboles, así como 4 opciones de profundidad máxima para cada árbol, 3 opciones para min_samples_split, 3 también para min_samples_leaf, y por último probar usando método de muestreo con reemplazo y sin reemplazo (True para reemplazo y False para sin reemplazo). Esto se puede observar en la siguiente configuración realizada:

```
# Usar grid search en el modelo elegido (Random Forest)
from sklearn.model_selection import GridSearchCV

# Usar varios hiperparámetros para ver con cuáles funciona mejor el modelo
param_grid = {
    'n_estimators': [100, 200, 300],          # Cantidad de árboles
    'max_depth': [10, 20, 30, None],          # Profundidad máxima de cada árbol
    'min_samples_split': [2, 5, 10],          # Número mínimo de muestras (samples) necesarias para dividir un nodo
    'min_samples_leaf': [1, 2, 4],            # Número mínimo de muestras necesarias en una hoja
    'bootstrap': [True, False]                # Método de muestreo: con (True) o sin (False) reemplazo
}

# Usar Grid Search
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Entrenar el modelo con los mejores hiperparámetros encontrados por el Grid Search
grid_search.fit(X_train, y_train)
```

Posteriormente, evalué este modelo mejorado:

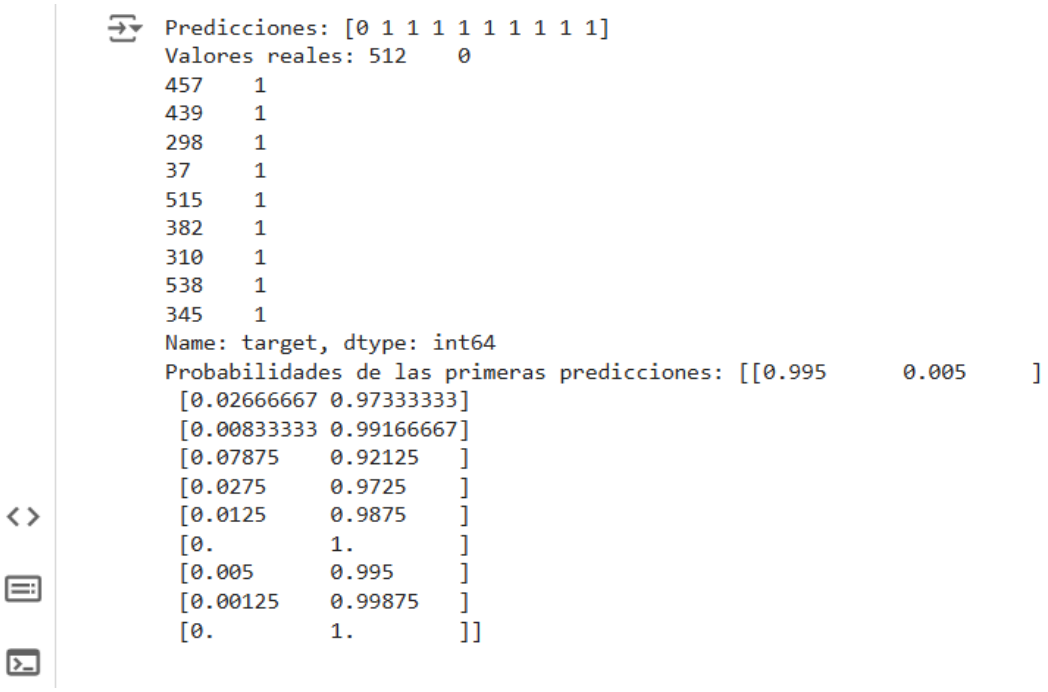
```
# Evaluar el mejor modelo
best_rf = grid_search.best_estimator_
score = best_rf.score(X_test, y_test)

# Comparar la exactitud obtenida con Grid Search vs la de antes (pre-Grid Search vs post-Grid Search)
print("Score del modelo antes de usar Grid Search:", acc_rf)
print("Score del modelo con los mejores hiperparámetros (después de usar Grid Search):", score)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Mejores hiperparámetros encontrados: {'bootstrap': False, 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}
Score del modelo antes de usar Grid Search: 0.956140350877193
Score del modelo con los mejores hiperparámetros (después de usar Grid Search): 0.9649122807017544
```

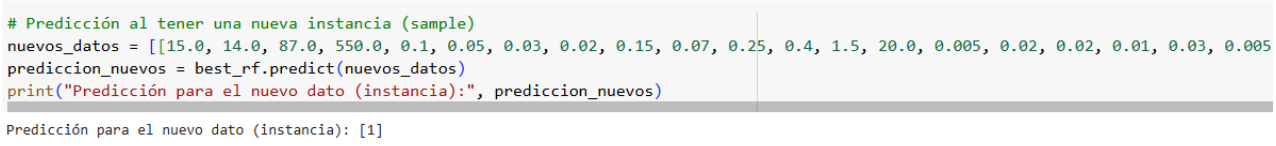
✓ 26s completed at 11:30 PM

Ahora que ya utilicé Grid Search en el modelo elegido, se puede observar que su rendimiento sí mejoró, pues antes del Grid Search tenía un score de 0.9561 y ahora aumentó a 0.9649, por lo cual se confirma que sí hubo mejora en el rendimiento tras haber utilizado Grid Search. Por lo tanto, después de eso procedí a ponerlo a prueba haciendo algunas predicciones con esta versión mejorada del modelo, tal como se puede observar a continuación:



```
Predicciones: [0 1 1 1 1 1 1 1 1 1]
Valores reales: 512    0
457    1
439    1
298    1
37     1
515    1
382    1
310    1
538    1
345    1
Name: target, dtype: int64
Probabilidades de las primeras predicciones: [[0.995      0.005     ]
 [0.02666667 0.97333333]
 [0.00833333 0.99166667]
 [0.07875    0.92125   ]
 [0.0275     0.9725    ]
 [0.0125     0.9875    ]
 [0.         1.        ]
 [0.005      0.995     ]
 [0.00125    0.99875   ]
 [0.         1.        ]]
```

Como se puede observar, este modelo mejorado básicamente acertó las 10 predicciones que lo puse a hacer, lo cual confirma que tuvo un buen rendimiento tal como se planeó, pues sus valores de predicción coinciden con los reales: un cero (o sea benigno) en la primera predicción, y un 1 (o sea maligno) en las 9 predicciones restantes. Asimismo, como funcionalidad adicional, el modelo puede también intentar predecir probabilidades sobre predicciones. Por último, aquí se puede observar cómo el modelo hace una predicción al considerar una nueva instancia de dato (nueva sample) haciendo uso de los mejores hiperparámetros encontrados:



```
# Predicción al tener una nueva instancia (sample)
nuevos_datos = [[15.0, 14.0, 87.0, 550.0, 0.1, 0.05, 0.03, 0.02, 0.15, 0.07, 0.25, 0.4, 1.5, 20.0, 0.005, 0.02, 0.02, 0.01, 0.03, 0.005
prediccion_nuevos = best_rf.predict(nuevos_datos)
print("Predicción para el nuevo dato (instancia):", prediccion_nuevos)
```

```
Predicción para el nuevo dato (instancia): [1]
```

Una vez hecho esto, procedí con el diagnóstico del grado de sesgo. En el caso del modelo de Random Forest, dado las métricas obtenidas de exactitud, precisión, recall, F1 y MSE, se puede observar que el rendimiento del modelo es alto (hay bajo error), lo cual significa que el modelo se adapta bien a los datos de entrenamiento, o sea que en este caso el grado de sesgo es bajo. A continuación, se puede observar esto más a detalle:

```
print(f"Sesgo (Bias) - Accuracy en entrenamiento: {accuracy_train}")
print(f"Sesgo (Bias) - Precision en entrenamiento: {precision_train}")
print(f"Sesgo (Bias) - Recall en entrenamiento: {recall_train}")
print(f"Sesgo (Bias) - F1 en entrenamiento: {f1_train}")
print(f"Sesgo (Bias) - MSE en entrenamiento: {mse_train}")
```

```
Sesgo (Bias) - Accuracy en entrenamiento: 1.0
Sesgo (Bias) - Precision en entrenamiento: 1.0
Sesgo (Bias) - Recall en entrenamiento: 1.0
Sesgo (Bias) - F1 en entrenamiento: 1.0
Sesgo (Bias) - MSE en entrenamiento: 0.0
```

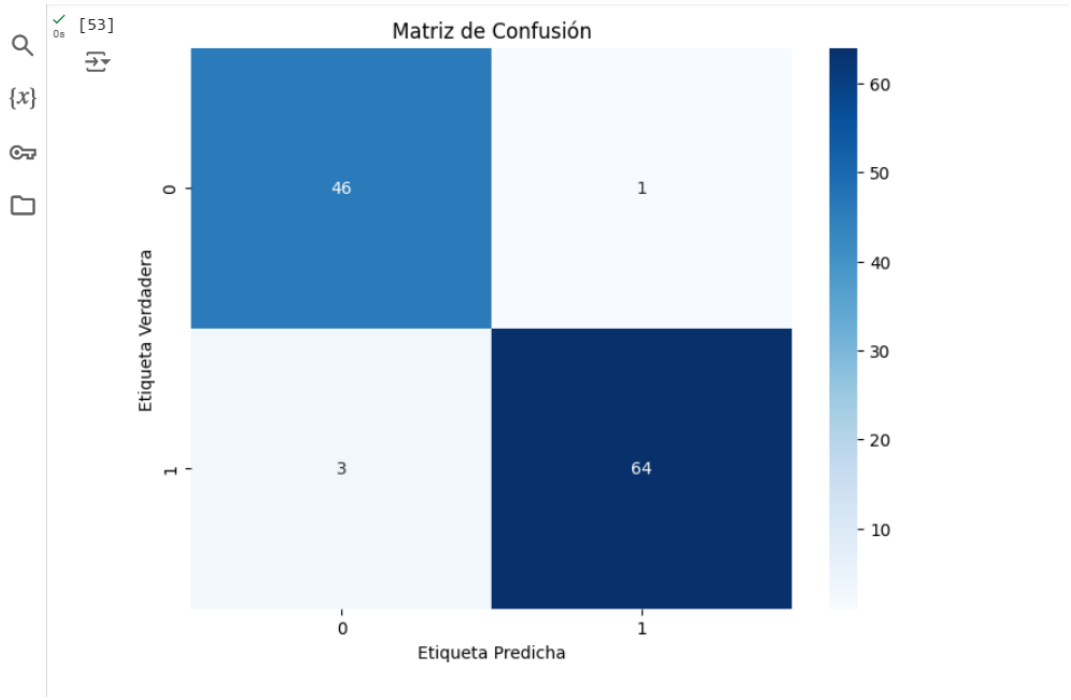
Para el diagnóstico sobre el grado de varianza me basé en lo siguiente: en los casos en donde un modelo tiene un buen rendimiento en los datos de entrenamiento pero tiene bajo rendimiento en los datos de prueba (overfitting), esto significa que tiene un grado de varianza alto; pero en este caso el modelo de Random Forest tuvo un buen desempeño tanto en entrenamiento como en prueba, pues su rendimiento no fue muy distante entre entrenamiento y prueba, lo cual indica que el grado de varianza en este caso es bajo. A continuación, se puede observar esta diferencia más a detalle:

```
[52] # Comparación entre entrenamiento y prueba
     print(f"Score en entrenamiento: {accuracy_train}")
     print(f"Score en prueba: {score}")
     print(f"Diferencia entre entrenamiento y prueba: {accuracy_train - score}")
     print(f"MSE en entrenamiento: {mse_train}")
     print(f"MSE en prueba: {mse_rf}")
     print(f"Diferencia entre entrenamiento y prueba en MSE: {mse_rf - mse_train}")
```

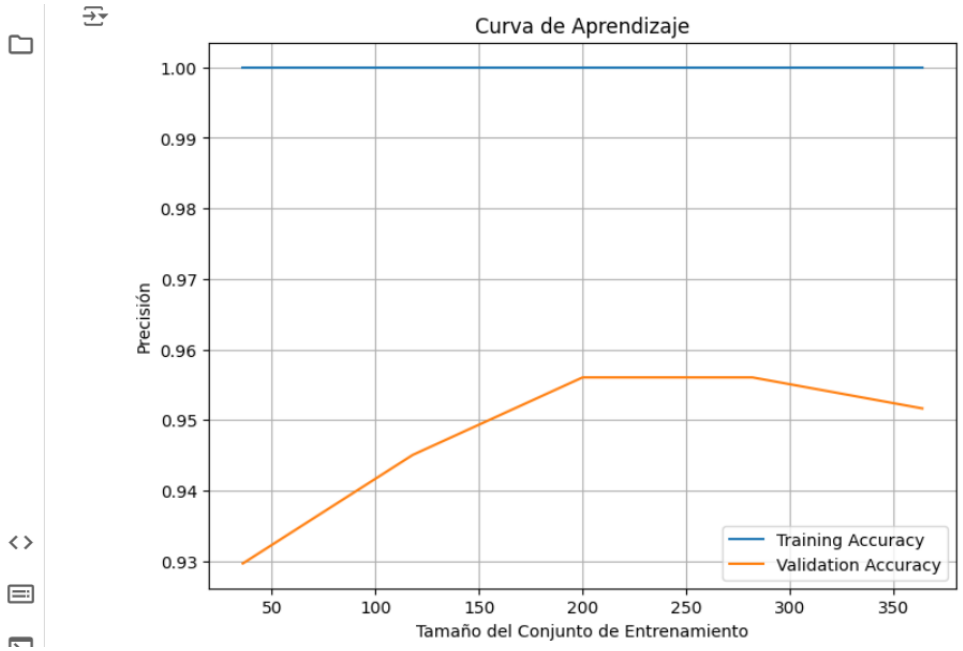
```
Score en entrenamiento: 1.0
Score en prueba: 0.9649122807017544
Diferencia entre entrenamiento y prueba: 0.03508771929824561
MSE en entrenamiento: 0.0
MSE en prueba: 0.043859649122807015
Diferencia entre entrenamiento y prueba en MSE: 0.043859649122807015
```

Con respecto al grado de ajuste, en este caso el grado de ajuste más adecuado se podría decir que es fitting (no hay overfitting pero tampoco hay underfitting), ya que tiene grado de sesgo bajo y grado de varianza bajo también, pues tiene error bajo tanto en entrenamiento como en prueba y la diferencia entre ambas no es grande, tal como se puede observar en las imágenes previas sobre el sesgo y la varianza.

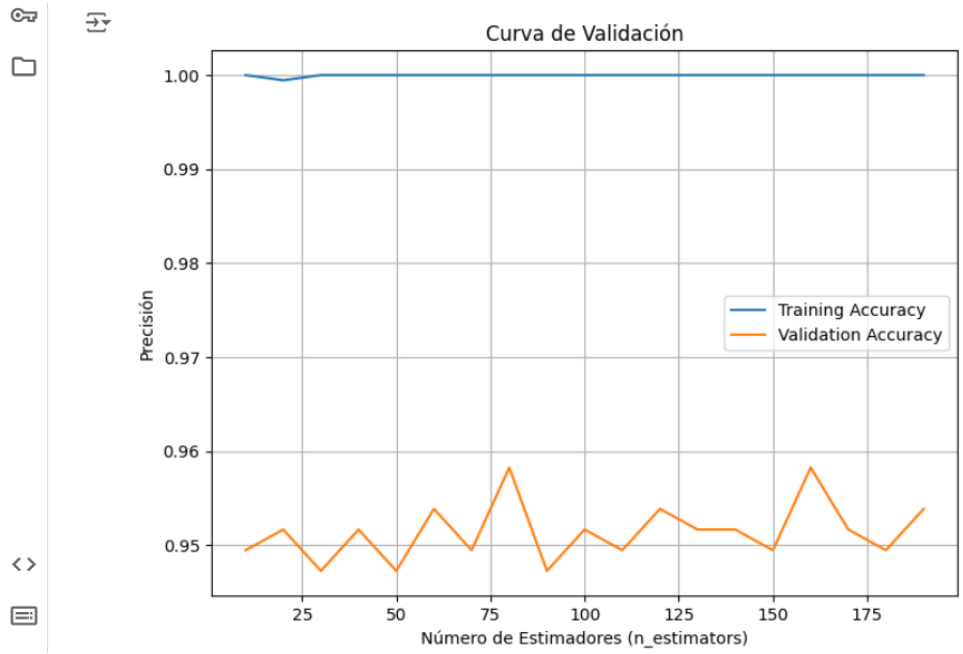
Finalmente, agregué 3 gráficas en el código, una sobre el desempeño del modelo conforme a la matriz de confusión, otra sobre la curva de aprendizaje del modelo, y la última sobre la curva de validación. Esto con el fin de poder apreciar esto de manera un poco más gráfica, tal como se puede observar a continuación:



En esta gráfica se pudo observar cómo se desempeñó el modelo tomando como base los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Estos valores de dicha matriz sirven como base para las métricas que se obtuvieron al evaluar el rendimiento del modelo con respecto a los resultados de exactitud, precisión, recall y score F1.



En esta gráfica se pudo apreciar de manera más gráfica la curva de aprendizaje del modelo de Random Forest, pudiendo observar una comparación entre la precisión de entrenamiento y validación.



Por último, en esta gráfica, se pudo observar la curva de validación del modelo, donde se puede ver una comparación entre la precisión de entrenamiento y validación.