



# Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Estado de México

**Materia:**

Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)

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

Santiago Espinosa Domínguez

A01747478

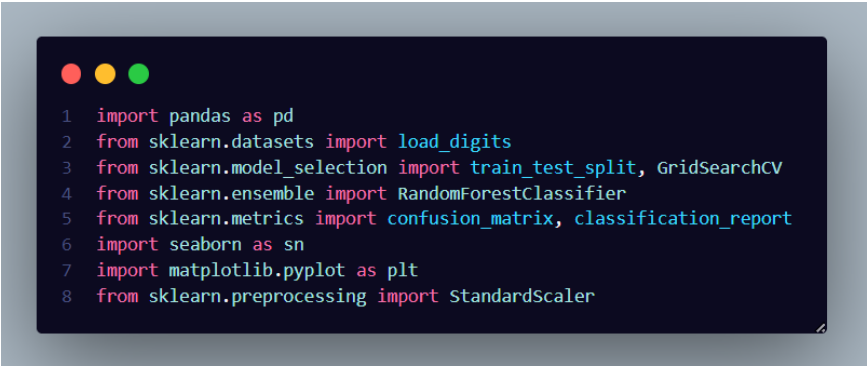
Jorge Adolfo Ramírez Uresti

9 de septiembre del 2024

El aprendizaje máquina que decidí usar es el modelo de **Random Forest**, que se basa en la creación de múltiples árboles de decisión durante el entrenamiento. Cada árbol se entrena con un subconjunto aleatorio de los datos y de las características, lo que ayuda a reducir el sobreajuste y a mejorar la precisión.

El modelo se basa en la combinación de varios modelos “débiles” (árboles de decisión) para formar un modelo más robusto y preciso. Al promediar o votar entre los resultados de todos los árboles, el modelo final es capaz de realizar predicciones más fiables y manejar mejor la variabilidad y los datos ruidosos.

Para lograr esto el primer paso fue importar las librerías necesarias para la creación del modelo, el Dataset, entre otras librerías importantes para su ajuste.

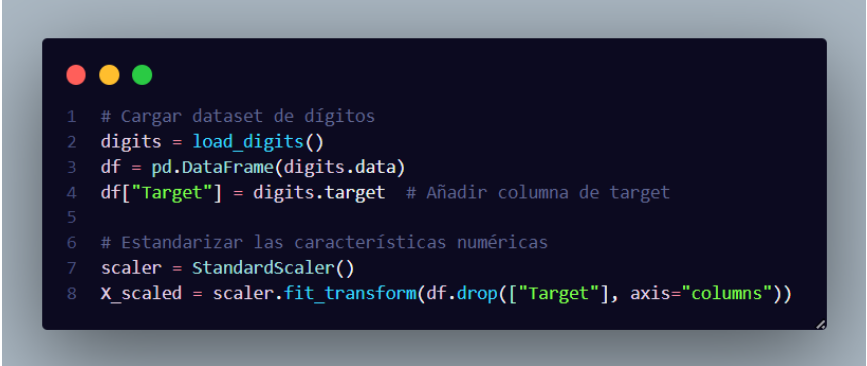


```
1 import pandas as pd
2 from sklearn.datasets import load_digits
3 from sklearn.model_selection import train_test_split, GridSearchCV
4 from sklearn.ensemble import RandomForestClassifier
5 from sklearn.metrics import confusion_matrix, classification_report
6 import seaborn as sn
7 import matplotlib.pyplot as plt
8 from sklearn.preprocessing import StandardScaler
```

Importación de librerías

Luego cargamos el Dataset de dígitos que se compone de un conjunto de datos que contiene imágenes de dígitos escritos a mano (del 0 al 9). Cada imagen está representada como una matriz de 8x8 píxeles, donde cada valor indica la intensidad del color en escala de grises. En total, el Dataset incluye 1,797 muestras, con 64 características por muestra (una por cada píxel) y una etiqueta asociada que indica el dígito correspondiente.

Después de cargar el Dataset lo agregamos a un DataFrame y escalamos los datos con la función de *StandardScaler*. El **escalado** iguala el rango de las características, mejorando el rendimiento y la estabilidad del modelo.



```
1 # Cargar dataset de dígitos
2 digits = load_digits()
3 df = pd.DataFrame(digits.data)
4 df["Target"] = digits.target # Añadir columna de target
5
6 # Estandarizar las características numéricas
7 scaler = StandardScaler()
8 x_scaled = scaler.fit_transform(df.drop(["Target"], axis="columns"))
```

Carga y estandarización de los datos

Luego de estandarizar los datos, los dividimos en tres conjuntos: **entrenamiento (60%)**, **validación (20%)**, y **prueba (20%)**. El mayor porcentaje se asigna al conjunto de entrenamiento para asegurar que el modelo tenga suficientes datos para aprender. El 20% de validación se utiliza para ajustar el modelo y evaluar su rendimiento con datos que no ha visto, permitiendo optimizar los hiperparámetros. Finalmente, el conjunto de prueba, también un 20%, se emplea para verificar qué tan bien generaliza el modelo después de haber sido ajustado

```
1 # Dividir los datos en entrenamiento (60%), validación (20%) y prueba (20%)
2 X_train_full, X_test, Y_train_full, Y_test = train_test_split(
3     X_scaled, digits.target, test_size=0.2, random_state=42
4 )
5 X_train, X_val, Y_train, Y_val = train_test_split(
6     X_train_full, Y_train_full, test_size=0.25, random_state=42 # 0.25 * 0.8 = 0.2
7 )
```

División de los datos

Luego para poder obtener el mejor modelo utilizamos la técnica de Grid Search ya que es utilizada para encontrar la mejor combinación de hiperparámetros en un modelo de aprendizaje automático, los hiper parámetros a evaluar son los siguientes:

- 'n\_estimators': Número de árboles en el bosque (50, 100 o 150).
- 'criterion': Criterio para medir la calidad de una división (gini o entropy).
- 'max\_depth': Profundidad máxima de los árboles (sin límite o hasta 10 o 20 niveles).

Finalmente buscamos los mejores hiperparámetros y entrenamos el modelo y lo probamos con los datos de validación.

```
1 # Definir hiperparámetros para GridSearchCV
2 param_grid = {
3     'n_estimators': [50, 100, 150],
4     'criterion': ['gini', 'entropy'],
5     'max_depth': [None, 10, 20]
6 }
7
8 # Optimización de hiperparámetros con el conjunto de validación
9 grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
10 grid_search.fit(X_train, Y_train)
11
12 # Evaluar el modelo en el conjunto de validación
13 print(f"Precisión en validación: {grid_search.score(X_val, Y_val)}")
```

Obtención del mejor modelo para el conjunto de validación

Al evaluar con los datos de validación obtenemos el mejor modelo, luego concatenamos los datos de entrenamiento y validación para volverlos a entrenar para finalmente volverlos a probar con el conjunto de prueba y mostrar el resultado.

```
1 # Usar el mejor modelo encontrado
2 best_model = grid_search.best_estimator_
3
4 # Entrenar nuevamente el mejor modelo con datos de entrenamiento y validación combinados
5 X_train_combined = pd.concat([pd.DataFrame(X_train), pd.DataFrame(X_val)])
6 Y_train_combined = pd.concat([pd.Series(Y_train), pd.Series(Y_val)])
7
8 best_model.fit(X_train_combined, Y_train_combined)
9
10 # Evaluar el modelo final en el conjunto de prueba
11 y_predicted = best_model.predict(X_test)
12
13 print(f"Precisión en prueba: {best_model.score(X_test, Y_test)}")
```

Obtención del mejor modelo para el conjunto de prueba

Finalmente, para obtener otra mejor visualización de los datos se crea una matriz de confusión en la que se compara los datos predichos con los datos reales del conjunto de prueba.

```
1 # Matriz de confusión
2 cm = confusion_matrix(Y_test, y_predicted)
3
4 # Reporte de clasificación
5 print("Reporte de clasificación:\n", classification_report(Y_test, y_predicted))
6
7 # Visualizar la matriz de confusión
8 plt.figure(figsize=(10,7))
9 sn.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True, xticklabels=digits.target_names, yticklabels=digits.target_names)
10 plt.xlabel("Predicciones")
11 plt.ylabel("Valores Reales")
12 plt.title("Matriz de Confusión")
13 plt.show()
```

Obtención de la matriz de confusión

Los resultados muestran que la precisión con el conjunto de validación alcanza un 98.61%, indicando un excelente desempeño durante la fase de ajuste y selección de hiperparámetros. De igual manera, la precisión en el conjunto de prueba es de 97.78%, demostrando que el modelo predice de manera correctamente con nuevos datos.

El reporte de clasificación muestra que el modelo tiene un desempeño destacado en la identificación de los dígitos, alcanzando una precisión y recall muy altos con una calificación arriba del 94% para varios dígitos. El puntaje F1 también refleja un equilibrio

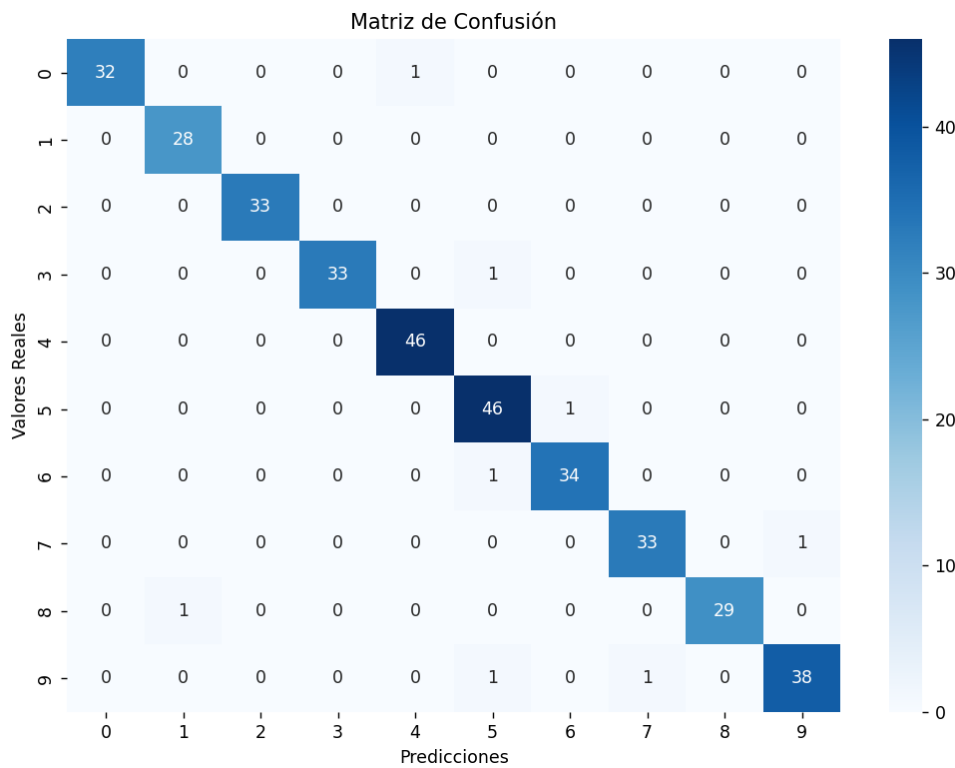
perfecto para estos dígitos, alcanzando casi el máximo en la mayoría de los dígitos. Estas métricas indican una alta efectividad y confiabilidad del modelo.

```
Precisión en validación: 0.9861111111111112
Precisión en prueba: 0.9777777777777777
Reporte de clasificación:
```

	precision	recall	f1-score	support
0	1.00	0.97	0.98	33
1	0.97	1.00	0.98	28
2	1.00	1.00	1.00	33
3	1.00	0.97	0.99	34
4	0.98	1.00	0.99	46
5	0.94	0.98	0.96	47
6	0.97	0.97	0.97	35
7	0.97	0.97	0.97	34
8	1.00	0.97	0.98	30
9	0.97	0.95	0.96	40
accuracy			0.98	360
macro avg	0.98	0.98	0.98	360
weighted avg	0.98	0.98	0.98	360

Métricas del modelo

La matriz de confusión respalda las métricas anteriores, ya que se observa que casi no se equivoco el modelo en predecir los dígitos.



Matriz de confusión