Maestría en Ciencia de Datos e Inteligencia Artificial

Tarea 02

Applied high performance computing

Informe Comparativo de Escalabilidad de Random Forest en Particiones SLURM: Debug vs Standard

Profesor:
Jose Antonio Fiestas Iquira

Integrantes:
Morocho Caballero, Rodolfo
Ramirez Martel, Max Houston
Velasquez Santos, Alberto Valentin

.

Resumen

Este informe presenta un análisis de rendimiento y escalabilidad de un clasificador Random Forest (Bosque Aleatorio) implementado con scikit-learn sobre un conjunto de datos sintéticos, variando el número de hilos de procesamiento (P) y el tamaño de la muestra (N). Los resultados demuestran una aceleración significativa en el tiempo de ajuste (T_{fit}) con el aumento de P, aunque con una eficiencia que disminuye progresivamente debido a la sobrecarga paralela. También se observa que el tiempo de ajuste escala de forma super-lineal con el tamaño de la muestra, lo cual es consistente con la complejidad algorítmica esperada. La exactitud del modelo se mantuvo alta y estable en todas las ejecuciones.

1. Introducción

Este informe analiza el rendimiento del entrenamiento de un clasificador **Random Forest** en función del número de hilos de procesamiento (P) y el tamaño de la muestra (N), comparando los resultados obtenidos en las particiones de **SLURM** `debug` y `standard`.

2. Configuración del Experimento

El experimento se ejecutó sobre una plataforma de Cómputo de Alto Rendimiento (HPC), utilizando el sistema de colas SLURM (script `rf.sh`). Se realizaron pruebas variando dos parámetros clave:

- **Número de Hilos** (P o `n_jobs`): $P \in \{1, 2, 4, 8, 16, 32\}$
- **Tamaño de la Muestra** (N o `n_samples`): $N \in \{100\text{k}, 200\text{k}, 400\text{k}, 800\text{k}\}$

Todos los demás hiperparámetros del Random Forest se mantuvieron fijos: `n_estimators` = 400, `max_depth` = 20, `n_features` = 40. Los tiempos reportados son el promedio de `reps` = 3 ejecuciones.

3. Análisis de Resultados entre Particiones

3.1. Comparativa de Tiempos de Entrenamiento y Particiones

La Tabla 1 compara los tiempos de entrenamiento promedio (`fit_time_s_avg`) entre las particiones `debug` y `standard`.

Cuadro 1: Tiempos Promedio de Entrenamiento (fit_time_s_avg) en segundos

N (n_samples)	P (n_jobs)	Debug (s)	Standard (s)	Diferencia Absoluta (s)
100 000	1	274.3691	273.3223	1.0468
100 000	2	148.8030	141.1416	7.6614
100 000	4	76.5603	76.4074	0.1529
100 000	8	42.2916	42.7293	-0.4377
100 000	16	24.6369	24.1222	0.5147
100 000	32	18.8874	17.8188	1.0686
200 000	1	—	597.3964	N/A
200 000	2	326.5193	310.9889	15.5304
200 000	4	170.8021	167.2803	3.5218
200 000	8	94.0955	91.0158	3.0797
200 000	16	54.5251	53.9700	0.5551
200 000	32	40.5150	39.0745	1.4405
400 000	4	373.6588	364.0706	9.5882
400 000	8	207.6706	212.3651	-4.6945
400 000	16	124.0623	122.0659	1.9964
400 000	32	90.2325	86.4191	3.8134
800 000	8	478.4954	492.0175	-13.5221

Cuadro 1: Tiempos Promedio de Entrenamiento (fit_time_s_avg) en segundos (Continuación)

N (n_samples)	P (n_jobs)	Debug (s)	Standard (s)	Diferencia Absoluta (s)
800 000	16	284.5556	281.8692	2.6864
800 000	32	194.1013	194.2279	-0.1266

Observaciones de Particiones: El rendimiento de cómputo es **similar** entre `debug` y `standard` (con `standard` siendo marginalmente más rápida en la mayoría de los casos). Esto implica que las ventajas de cada partición son de naturaleza administrativa (cola, límites de tiempo), no de rendimiento bruto del hardware.

4. Análisis de Escalabilidad por Hilos (P)

4.1. Figura 1: Tiempo de Ajuste vs. Número de Hilos

La Figura 1 muestra la reducción del tiempo de entrenamiento al aumentar el número de hilos P (escala logarítmica en el eje Y), lo que confirma que el algoritmo Random Forest es altamente paralelizable.

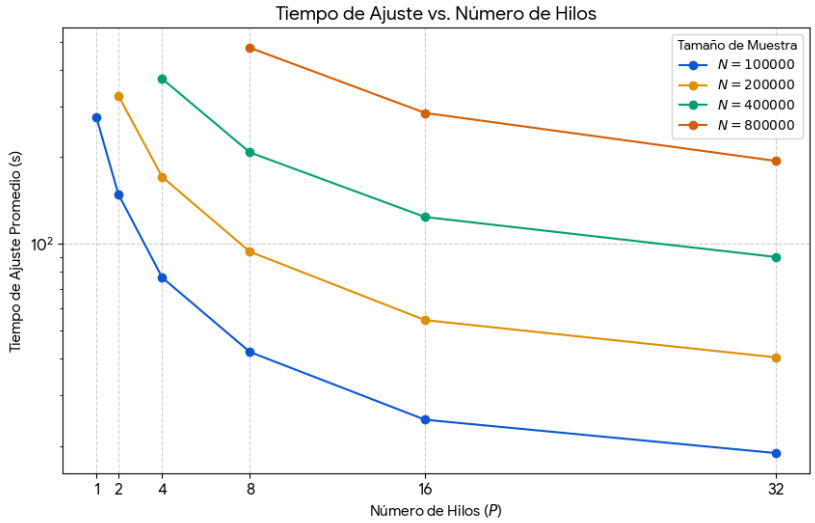

Figura 1: Figura 1: Tiempo de Ajuste Promedio vs. Número de Hilos (P).

Conclusiones de la Figura 1 (Tiempo Absoluto):

1. **Ley de Amdahl Evidente:** Se observa una fuerte disminución de tiempo inicialmente (de $P = 1$ a $P = 8$). La reducción se hace **menos pronunciada** de $P = 16$ a $P = 32$, especialmente para los problemas más grandes ($N = 400\,000$ y $N = 800\,000$), lo que indica que el **overhead** y la parte secuencial del código (**Ley de Amdahl**) comienzan a dominar.

2. **Complejidad por Tamaño (N):** La separación vertical entre las curvas muestra que el tiempo de ejecución escala aproximadamente de forma lineal con el tamaño de la muestra N (ej., $N = 200\,000$ toma aproximadamente el doble de tiempo que $N = 100\,000$ para el mismo P).

4.2. Figura 2: Aceleración Relativa de Ajuste vs. Número de Hilos

La Figura 2 muestra el **Speedup** con diferentes bases de referencia (P_{base}), lo que revela cómo la saturación del paralelismo impacta la eficiencia en función del tamaño del problema N.

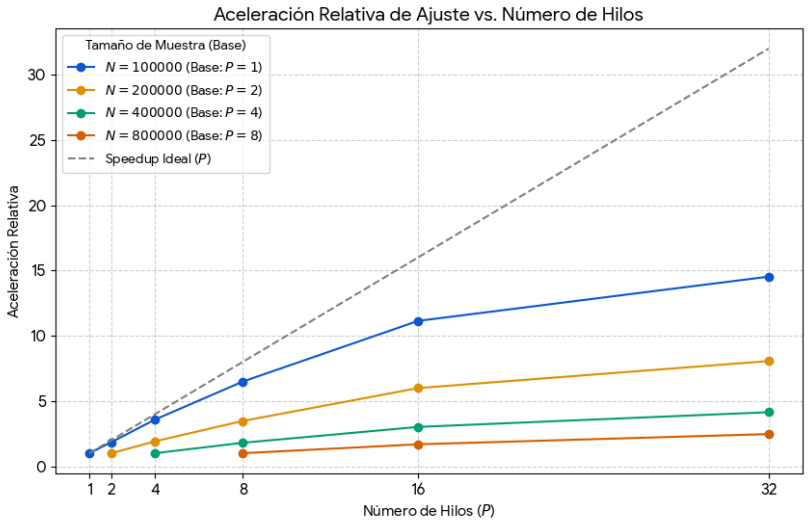

Figura 2: Figura 2: Aceleración Relativa ($S_{rel}(P)$) vs. Número de Hilos (P).

Conclusiones de la Figura 2 (Aceleración Relativa):

1. **Mejor Escalabilidad Fuerte ($N = 100\,000$):** La curva azul (Base $P = 1$) es la única que se acerca a la línea de Speedup Ideal, alcanzando casi $15\times$ en $P = 32$. Esto confirma que, para problemas con carga de trabajo pequeña, el sistema puede gestionar el paralelismo eficientemente.

2. **Saturación con Problemas Grandes:** Para $N = 800\,000$ (Base $P = 8$, curva marrón), el Speedup es el más bajo ($\approx 2{,}5\times$ en $P = 32$). Esto significa que al pasar de 8 a 32 hilos, la ganancia de tiempo es muy pequeña. El problema ya está casi completamente saturado de paralelismo en $P = 8$, y el aumento de hilos a partir de ahí solo aumenta el costo de **overhead**.

3. **Rendimientos Decrecientes:** Ambas gráficas demuestran **rendimientos decrecientes** al aumentar P. Las curvas de la Figura 1 se aplanan, y las curvas de la Figura 2 se separan significativamente de la línea ideal. En la práctica, utilizar más de $P = 16$ ofrece una ganancia marginal que podría no justificar el uso de tantos recursos.

5. Ventajas y Desventajas de las Particiones SLURM

La elección de la partición debe priorizar los objetivos del trabajo (iteración rápida vs. ejecución final), ya que el rendimiento de cómputo por núcleo es similar.

5.1. Partición `debug`

Ventajas:

- **Inicio Rápido:** Alta prioridad y baja latencia en la cola, esencial para **depuración** y **pruebas iniciales** (Validación de código).
- **Ciclo de Desarrollo Ágil:** Permite verificar rápidamente la configuración de recursos y la lógica del código sin largos tiempos de espera.

Desventajas:

- **Límites Estrictos de Recursos:** Imposición de un **tiempo máximo de pared** muy limitado (ej. 30 minutos).
- **Inadecuada para Producción:** No permite ejecutar problemas grandes (como el caso de $N = 800\,000$ con $P = 32$) o tareas que requieren muchas horas de cómputo.

5.2. Partición `standard`

Ventajas:

- **Producción y Larga Duración:** Diseñada para **ejecuciones finales** que requieren grandes asignaciones de tiempo y recursos.
- **Escalabilidad Realista:** Permite ejecutar el rango completo de problemas probados, proporcionando la base más adecuada para el **análisis de rendimiento formal** y producción.

Desventajas:

- **Mayor Tiempo de Espera en Cola:** Los trabajos suelen tener una **menor prioridad**, lo que resulta en tiempos de espera significativos antes de la ejecución.
- **Ineficiente para Debugging:** Su uso es ineficiente para el desarrollo debido a la interrupción del ciclo de prueba-error por los tiempos de espera.