Implementing Parallel Programming in the Machine Learning Model Validation Process

Cristian Penagos
Ingeniería de Sistemas
Universidad de Antioquia
cristian.penagos@udea.edu.co

Daniel Burbano
Ingeniería de Sistemas
Universidad de Antioquia
esteban.burbano@udea.edu.co

Luis Velásquez
Ingeniería de Sistemas
Universidad de Antioquia
luis.velasquezg@udea.edu.co

Mateo Rivera
Ingeniería de Sistemas
Universidad de Antioquia
mateo.rivera@udea.edu.co

***Abstract*—En esta investigación nos enfocaremos en implementar computación paralela sobre la técnica de evaluación de modelos de inteligencia artificial conocida como 'validación cruzada con K-Folds', con el objetivo de determinar el efecto del paralelismo en el rendimiento del proceso de validación.**

***Index Terms*—paralelismo, programación paralela, validación cruzada, k-fold, modelo knn, aprendizaje automático, machine learning, inteligencia artificial**

I. Introducción

El reto identificado se centra en la optimización del rendimiento en el ámbito computacional, específicamente en el contexto del Machine Learning. Para ello, se hará uso de la técnica del paralelismo buscando mejorar el desempeño, en términos de tiempo, del proceso de validación en un modelo de Machine Learning.

La programación paralela es de gran importancia en el contexto tecnológico actual ya que ayuda, entre otras cosas, al aprovechamiento del hardware multinúcleo, con el que cuenta la mayoría de las computadoras modernas, posibilitando la ejecución simultánea de tareas o cálculos computacionales. Además, el paralelismo ayuda significativamente en el procesamiento y análisis de grandes volúmenes de datos, habituales en proyectos de Big Data y de Machine Learning, por nombrar algunos, haciendo posible que el tiempo de ejecución sea reducido al distribuir la carga computacional entre diferentes máquinas o procesadores.

En este proyecto, aplicamos paralelismo al proceso de validación de un modelo KNN de Machine Learning de clasificación, más concretamente al método de validación cruzada K-Folds, haciendo uso de la librería *Multiprocessing* de Python.

II. Marco Teórico

Dentro de los elementos teóricos más relevantes para entender el reto se encuentra el proceso del Machine Learning, especialmente el proceso de validación de un modelo usando validación-cruzada con K-Folds, el modelo KNN, el paralelismo y el manejo de la librería Multiprocessing de Python, la cual usamos para implementar el paralelismo.

A. Machine Learning

Es una rama de la *Inteligencia Artificial* que consiste de un conjunto de técnicas que permite identificar patrones a partir de ejemplos, dichos ejemplos están contenidos en una base de datos o *dataset*. Estos patrones identificados le permitirán a la máquina poder hacer predicciones, o clasificaciones, sobre muestras que no conoce, lo que se podría considerar como "aprendizaje", de allí el nombre de "*Machine Learning*" (o *Aprendizaje Automático*, en español).

Amazon Web Services, Inc., define el Machine Learning de la siguiente forma:

> "Ciencia de desarrollo de algoritmos y modelos estadísticos que utilizan los sistemas de computación con el fin de llevar a cabo tareas sin instrucciones explícitas, basándose en patrones e inferencias. Los sistemas de computación utilizan algoritmos de Machine Learning para procesar grandes cantidades de datos históricos e identificar patrones de datos. Esto les permite generar resultados con mayor precisión a partir de un conjunto de datos de entrada. Por ejemplo, los científicos de datos pueden entrenar una aplicación médica para diagnosticar el cáncer con imágenes de rayos X a partir del almacenamiento de millones de imágenes escaneadas y diagnósticos correspondientes." [1]

Un *modelo* de Machine Learning es una representación matemática de un proceso o sistema basado en datos. Esta representación matemática contiene *parámetros* que deben ser ajustados para que se puedan realizar predicciones o clasificaciones confiables de nuevas muestras. Es aquí donde entra el proceso de validación [2].

B. Proceso de Validación de un Modelo de Machine Learning

El *proceso de validación* es esencial para hacer una buena elección de los parámetros del modelo, ya que esto le permitirá al modelo hacer predicciones con un alto grado de confiabilidad. Para ello, se utiliza una metodología de validación que nos permitirá conocer el comportamiento del

sistema con respecto a muestras que desconoce y, basándonos en ello, poder seleccionar los mejores parámetros.

Las metodologías de validación más utilizadas son *Validación Cruzada* y *Bootstrapping* [3]. Nos enfocaremos en la primera; la validación cruzada con k-fold.

La *validación cruzada* (o *k-fold cross-validation*) consiste en dividir el conjunto de datos, o *dataset*, en dos partes: un conjunto de entrenamiento (*training*) y otro de prueba (*test*). Por lo general, los conjuntos de entrenamiento y de prueba están conformados por el 80% y 20% del dataset, respectivamente [3].

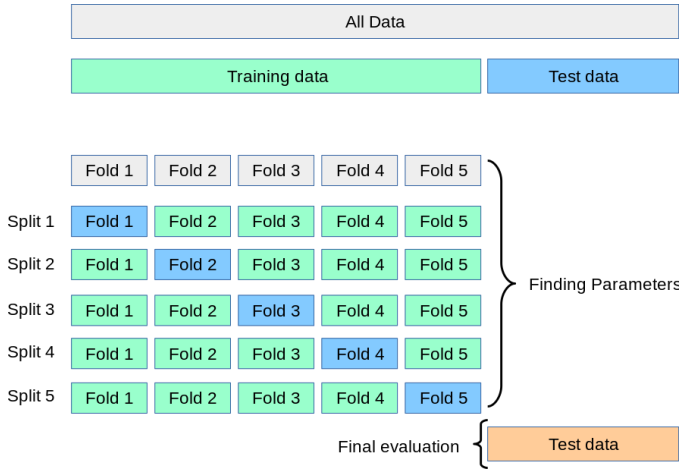

Fig. 1. Ejemplo de validación cruzada K-Fold con k=5.

A su vez, el conjunto de entrenamiento (*Training data*) se divide aleatoriamente en k subconjuntos disjuntos a los que le llaman "folds". Se usarán $k - 1$ subconjuntos para entrenar el modelo y el subconjunto restante se usará para validarlo.

El proceso anterior se repite por cada subconjunto k y el error se cuantifica como el promedio del error obtenido en las k repeticiones [3], como se muestra en la figura 1.

C. Modelo de K-Vecinos Más Cercanos (KNN)

El modelo de *k-vecinos más cercanos* (o *K-Nearest Neighbors*, en inglés, o KNN) se trata de un modelo de Machine Learning que calcula las distancias de un punto de datos (muestra nueva), dentro de un espacio de características, con respecto a un número K predefinido de vecinos más cercanos para poder predecir o clasificar dicha muestra (representada como un punto). La idea principal de este algoritmo es que los puntos de datos similares tienden a agruparse en el espacio.

Así que, en el caso de un problema de clasificación, el modelo asignará al nuevo punto la etiqueta (o clasificación) que es más común entre los K vecinos más cercanos a él. De manera análoga, en el caso de la regresión (o predicción), el modelo calcula el promedio de los valores de los K vecinos más cercanos al punto nuevo y se lo asigna como valor predicho. En nuestro caso, nuestro problema de Machine Learning será de clasificación.

D. Paralelismo

El paralelismo analizado de forma general, se refiere a la técnica de dividir un problema en partes más pequeñas y resolver esas partes de forma simultánea. En lugar de ejecutar una tarea de principio a fin en una sola secuencia, el paralelismo permite realizar múltiples tareas al mismo tiempo, distribuyendo la carga de trabajo entre diferentes unidades de procesamiento. Tenemos dos tipos principales de paralelismo.

- Paralelismo de Datos: En este tipo de paralelismo, se divide un conjunto de datos en partes más pequeñas y se procesan esas partes de manera simultánea. Cada unidad de procesamiento trabaja con su propia porción de datos, lo que puede acelerar significativamente la ejecución de tareas que involucran grandes cantidades de información.

- Paralelismo de Tareas (o Paralelismo de Tareas Independientes): En este tipo de paralelismo, se dividen las tareas en sub-tareas más pequeñas e independientes que pueden ejecutarse simultáneamente. Cada tarea o sub-tarea se asigna a un procesador o hilo de ejecución separado. Esto es particularmente útil cuando hay tareas que no dependen entre sí y, por lo tanto, pueden ejecutarse al mismo tiempo sin conflicto.

E. Librería Multiprocessing de Python

Multiprocessing es un módulo de Python que nos permite aumentar la eficacia de los scripts asignando tareas a distintos procesos. Admite procesos de generación utilizando una API similar al módulo *threading*. La diferencia radica en que los procesos no comparten memoria con otros procesos, a diferencia de los hilos, los cuales comparten el espacio de memoria entre ellos [4][5].

Es una herramienta Ofrece concurrencia local y remota, evitando efectivamente el bloqueo global del intérprete mediante el uso de subprocesos en lugar de subprocesos. Debido a esto, el módulo multiprocessing permite al programador aprovechar al máximo varios procesadores en una máquina determinada.

El módulo multiprocessing también introduce API que no tienen análogos en el módulo threading. Un buen ejemplo de esto es el objeto Pool que ofrece un medio conveniente de paralelizar la ejecución de una función a través de múltiples valores de entrada, distribuyendo los datos de entrada a través de procesos (paralelismo de datos) [6].

El siguiente ejemplo demuestra la práctica común de definir tales funciones en un módulo para que los procesos secundarios puedan importar con éxito ese módulo. Este ejemplo básico de paralelismo de datos usando Pool:

```python
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

imprimirá la salida estándar

```
[1, 4, 9]
```

Tuberías (Pipes) y Colas (Queues)

Cuando se usan múltiples procesos, uno generalmente usa el paso de mensajes para la comunicación entre procesos y evita tener que usar primitivas de sincronización como locks.

Para pasar mensajes se puede usar tuberías (*pipe*) para una conexión entre dos procesos, o una cola (*queue*), que permite múltiples productores y consumidores.

III. Metodología

Se tienen n posibles configuraciones para el modelo que se desea evaluar, y k folds, por lo que en total se necesitarán entrenar y evaluar $k \times n$ modelos, así que proponemos dos formas de abordar la paralelización de la validación cruzada:

1. Realizar de forma secuencial la evaluación de los n modelos y aplicar la paralelización en los k folds.
2. Realizar de forma paralela la evaluación de los n modelos y ejecutar de forma secuencial los k fold.

Dado que usualmente $k \in \{5, 10\}$ y que la cantidad de configuraciones que podemos considerar de un modelo puede superar en órdenes de magnitud el valor que puede tomar k (Por ejemplo, en un perceptrón multicapa se pueden considerar diferentes combinaciones de cantidad de capas ocultas, cantidad de neuronas por capa y funciones de activación), se ha tomado la decisión de abordar este problema de la segunda forma. Para ello se ha tomado el modelo de clasificación $\varkappa$ vecinos más cercanos con 250 posibles configuraciones ($\varkappa$ toma valores impares desde 1 hasta 499 ambos incluídos), un dataset de 10,000 muestras con 10 clases para clasificar, lo que da un total de 250 * 10 =2,500 modelos a evaluar.

Así que, se desea tomar el tiempo de cuánto tarda realizar un computador la evaluación de estos 2,500 modelos usando *n_job* = 1, 2, …, 12 trabajos independientes; cabe destacar que cuando *n_job* = 1 se está considerando el caso sin paralelización.

Los resultados obtenidos en términos de tiempo pueden variar dependiendo de las características del equipo donde se desee replicar los experimentos realizados, en este caso se usó un computador portátil de 16 GB de RAM, con un procesador Intel Core i7-9750H, con sistema operativo Windows 11 Home, y tarjeta gráfica de video dedicada GTX 1660Ti; además se replicó el experimento dos veces:

1. Después de usar el computador en tareas de ofimática (Corrida 1).
2. Recién encendido el computador (Corrida 2)

En ambos casos se garantizó que el experimentador cerrara todas las pestañas, dejando solamente abierta la terminal de Windows PowerShell para la ejecución del programa, aumentando homogeneidad de condiciones durante el proceso de ejecución del programa.

Se usó la base de datos [7].

Nota: La letra "k" se usa a lo largo del texto haciendo referencia a "k-Folds" mientras que "$\varkappa$" al modelo de "K vecinos más cercanos".

IV. Implementación

Se crearon dos archivos con extensión ".py"

- `experiments.py`: Este archivo contiene las funciones que permiten realizar experimentaciones con y sin paralelización, nótese que la métrica calculada para evaluar los modelos es "Accuracy".
  La función `experiments_parallelism` es la encargada de materializar la paralelización dada la cantidad de subprocesos que se deseen crear, esta función es agnóstica al modelo, es decir, se desarrolló tal forma de que no se deba crear una función de paralelización por modelo en particular, es decir, una función para árboles de decisión, otra para $\varkappa$ vecinos más cercano, etc., ya que esta recibe un modelo preconfigurado con el cuál realiza las experimentaciones; esto se hizo debido a que cada modelo de machine learning tiene una cantidad variable de hiper parámetros: esta función retorna una lista de tripletas, donde cada tripleta contiene en ese orden.
  1. Los hiper parámetros usados en la experimentación
  2. La media aritmética del accuracy calculado en los "k-folds"
  3. La desviación estándar del accuracy calculado en los "k-folds"
- `main.py`: En este archivo se carga el dataset con el que se van a realizar las experimentaciones, se detallan cuáles son los hiper parámetros con los cuáles se van a experimentar, y la experimentación con las diferentes configuraciones de paralelización.

V. Resultados

En la figura 2, se ven los resultados obtenidos después de ejecutar el programa bajo las dos condiciones mencionadas en la metodología.

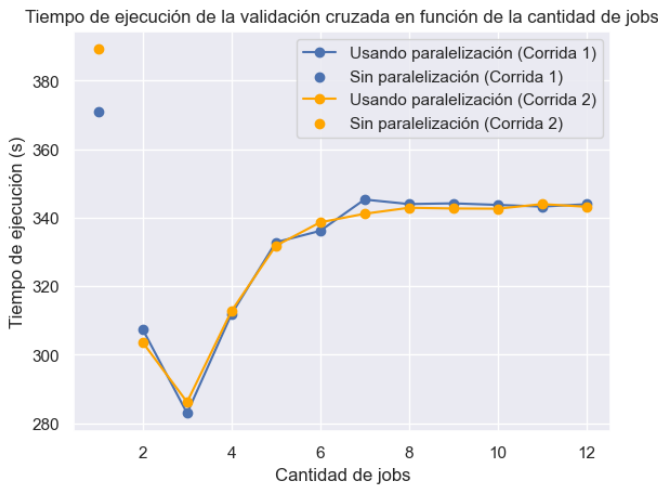

Fig. 2. Tiempo(s) que toma la evaluación de los 2,500 modelos en función de la cantidad de subprocesos.

Del gráfico podemos destacar los siguientes aspectos:

- La variación que existe entre las dos ejecuciones es poca, ya que casi que todos los puntos se solapan y ambas curvas se aproximan; además se ratifica el hecho de que no es cuestión de azar de que en *n_job* = 3 se dé la mínima reducción de tiempo.
- Intuitivamente se podría pensar que entre más subprocesos, hay más aprovechamiento de la paralelización, sin embargo, a partir de los 4 *jobs* el desempeño comienza a empeorar.
- Hay una convergencia en *n_job* = 7, a partir de este punto no hay ni mejora ni empeoramiento.
- Independientemente de la cantidad de subprocesos que tengamos siempre hay una mejora de desempeño con respecto a la ejecución secuencial.
- También se podría pensar que al ejecutar un programa que se demora x segundos, el tiempo que tomará realizarlo al paralelizarlo en *n_job* subprocesos es x/n_job, sin embargo, del gráfico se rescata que esto no es verdad, en el caso de *n_job* = 2 existe una mejora aproximada del 22%.

A continuación, se anexa la tabla 1, donde se detallan los resultados obtenidos en las dos ejecuciones de la experimentación:

n_job	**Ejecución 1**	**Ejecución 2**
Sin paralelizar	370.81	389.16
2	307.38	303.59
3	282.92	286.17
4	311.79	312.72
5	332.82	331.80
6	336.14	338.62
7	345.29	341.13
8	343.97	342.88
9	344.17	342.67
10	343.72	342.61
11	343.23	343.95
12	343.91	343.18

Tabla. 1. Tiempo(s) que toma la evaluación de los 2,500 modelos en función de la cantidad de subprocesos.

VI. Conclusiones

En conclusión, después de la experimentación realizada, podemos observar que el paralelismo en Machine Learning ofrece ventajas significativas en términos de velocidad, escalabilidad y eficiencia en el manejo de grandes conjuntos de datos.

Sin embargo, su implementación exitosa requiere de una planificación cuidadosa y la consideración de algunos desafíos asociados, entre los cuales se encuentra la sincronización de datos entre nodos, la gestión de la comunicación y la coordinación efectiva, estos son aspectos críticos que deben abordarse para aprovechar al máximo los beneficios del paralelismo.

Aunque si abordamos de manera eficiente estos aspectos críticos, siguiendo una metodología específica que apoye el logro de los objetivos, el paralelismo tiene el potencial para desempeñar un papel importante en el avance de tecnologías futuras, por lo que también se concluye que, dentro de la temática, se encuentra un campo de acción e investigación para conocer y explorar, de una importancia significativa, dejándonos como resultado una base sólida para avanzar en su exploración.

Referencias

[1] "What is Machine Learning? - Enterprise Machine Learning Explained - AWS" *Amazon Web Services, Inc.* https://aws.amazon.com/what-is/machine-learning/

[2] J. Arias, "Complejidad de modelos," *Introducción al Machine Learning*, 2021. https://jdariasl.github.io/ML_2020/Clase%2006%20-%20Complejidad%20de%20modelos%2C%20sobreajuste%20y%20metodolog%C3%ADas%20de%20validaci%C3%B3n.html (accessed Nov. 26, 2023).

[3] J. Arias, "Medidas de Error y Selección del Modelo: Validación," *Modelamiento y Simulación de Sistemas*, 2014.

[4] D. Chung, "Multiprocessing in Python," *Machine Learning Mastery*, Apr. 25, 2022. https://machinelearningmastery.com/multiprocessing-in-python/

[5] "Difference between Multiprocessing and Multithreading - GeeksforGeeks," *GeeksforGeeks*, May 22, 2019. https://www.geeksforgeeks.org/difference-between-multiprocessing-and-multithreading/

[6] Python Software Foundation, “Multiprocessing — Paralelismo basado en procesos,” *Python documentation*. https://docs.python.org/es/3/library/multiprocessing.html (accessed Nov. 26, 2023).
[7] DBDMG. (2019). MNIST Test Dataset [Conjunto de datos]. GitHub. https://raw.githubusercontent.com/dbdmg/data-science-lab/master/datasets/mnist_test.csv