Informe sobre la Implementación de Algoritmos de Machine Learning Secuenciales y Concurrentes en Go

Alumno: Henry Josue Diaz Huarcaya - U20201C579

Contexto

La programación concurrente es un paradigma clave en el desarrollo de sistemas modernos, donde la eficiencia y la escalabilidad son esenciales. En el contexto de machine learning, la capacidad de procesar grandes cantidades de datos de manera rápida y eficiente es crucial. Para aprovechar las ventajas de la concurrencia en Go, utilizamos goroutines, que permiten la ejecución de múltiples tareas en paralelo, mejorando el rendimiento y reduciendo los tiempos de ejecución, especialmente en datasets masivos.

Descripción de la Tarea

En este proyecto, se implementaron diversos algoritmos de machine learning tanto en versiones secuenciales como concurrentes utilizando goroutines en Go. Los algoritmos implementados son:

- Árboles de Decisión y Bosques Aleatorios (Random Forests)
- Máquinas de Soporte Vectorial (SVM)
- Redes Neuronales Artificiales y Redes Neuronales Profundas (Deep Learning)
- Filtrado Colaborativo y Modelos Basados en Factores Latentes

Cada algoritmo fue probado en datasets de gran tamaño, comparando el rendimiento de las versiones secuenciales con las concurrentes. El objetivo fue identificar mejoras en el uso de la concurrencia y analizar los resultados obtenidos.

Implementaciones

Árboles de Decisión

La implementación de Árboles de Decisión de forma secuencial presentó un tiempo de ejecución de **84.79ms**. En cambio, la versión concurrente, que entrenó cuatro modelos en paralelo, presentó tiempos de ejecución individuales para cada modelo, completando todo el proceso en **339.64ms**. Aunque la versión concurrente no fue significativamente más rápida, aprovechó mejor la paralelización en sistemas con múltiples núcleos.

Bosques Aleatorios (Random Forests)

El algoritmo de Bosques Aleatorios secuencial completó su ejecución en **0s** con una precisión del **51.50%**. La versión concurrente también tuvo un tiempo total de **0s**, pero la precisión disminuyó a **44.00%**. En este caso, la concurrencia no mostró una mejora significativa en términos de rendimiento de tiempo, y hubo una ligera degradación en la precisión del modelo.

Máquinas de Soporte Vectorial (SVM)

El entrenamiento secuencial de SVM sobre 1,000,000 registros tomó **132.97ms**. La versión concurrente completó la misma tarea en **270.76ms**, mostrando que, en este caso, la concurrencia no mejoró el tiempo de ejecución. Sin embargo, las diferencias en los pesos finales de los modelos sugieren que la división de trabajo pudo haber influido en la convergencia del modelo.

Redes Neuronales

Para las redes neuronales, el entrenamiento secuencial tomó **2.01s** mientras que la versión concurrente tardó **2.53s**, un incremento en el tiempo debido a la sobrecarga de coordinación entre goroutines. Sin embargo, esta implementación concurrente podría optimizarse aún más para aprovechar mejor la concurrencia en capas específicas de la red.

Deep Learning

El modelo de Deep Learning secuencial tuvo un tiempo de ejecución de **12.13ms**, mientras que el concurrente fue significativamente más rápido, con un tiempo de **2.09ms**. Aquí, la concurrencia demostró una mejora significativa, posiblemente debido a la paralelización de operaciones en diferentes capas de la red.

Filtrado Colaborativo

En el Filtrado Colaborativo, la versión secuencial completó la tarea en **3.19s**, mientras que la versión concurrente fue mucho más rápida, con un tiempo de **340.58ms**. Este es un claro ejemplo de cómo la concurrencia mejora drásticamente el rendimiento, al permitir que varias predicciones se realicen en paralelo.

Modelos Basados en Factores Latentes

Finalmente, en los modelos basados en factores latentes, tanto la versión secuencial como la concurrente obtuvieron una precisión del **0%**, indicando que el modelo no fue capaz de aprender de los datos. Sin embargo, la versión concurrente fue más lenta, con un tiempo de **4.18ms** frente a **1.04ms** de la versión secuencial.

Análisis de Resultados

El análisis de los resultados muestra que la concurrencia puede mejorar significativamente el rendimiento en ciertos tipos de algoritmos, como en el caso del Filtrado Colaborativo y Deep Learning, donde la ejecución concurrente fue considerablemente más rápida. Sin embargo, en otros casos, como SVM y Random Forests, la concurrencia no mejoró los tiempos de manera significativa e incluso disminuyó la precisión del modelo.

En resumen:

- **Ventajas**: La concurrencia es especialmente efectiva en algoritmos que pueden paralelizarse naturalmente, como las Redes Neuronales y el Filtrado Colaborativo.
- **Desventajas**: La concurrencia puede añadir una sobrecarga innecesaria en casos donde la secuencialidad del algoritmo sea más eficiente o cuando la coordinación entre goroutines no esté optimizada.

Conclusión

La implementación concurrente en Go usando goroutines ofrece ventajas significativas en términos de paralelización de tareas y optimización del uso de recursos. Sin embargo, no todos los algoritmos de machine learning se benefician de la concurrencia, y en algunos casos, el costo de coordinación entre goroutines puede superar los beneficios. Es importante analizar el algoritmo en cuestión para decidir si la concurrencia es una opción adecuada.