Universidad Pablo de Olavide

K-Nearest Neighbors

09/02/2024

Sistemas Distribuidos

"No temo a los ordenadores; lo que temo es quedarme sin ellos"

-Isaac Asimov

Grupo 11

Antonio Moreno Mendez

Pablo Sanchez Troncoso

Antonio Garrido Masse

Marcos Villarreal Muñoz

Índice

Descripción del problema y su solución

Se han tomado mediciones de la temperatura por hora de varios días. Dadas tales mediciones, se pretende predecir la temperatura por hora que se dará en los días siguientes a uno concreto.

Los datos de entrada

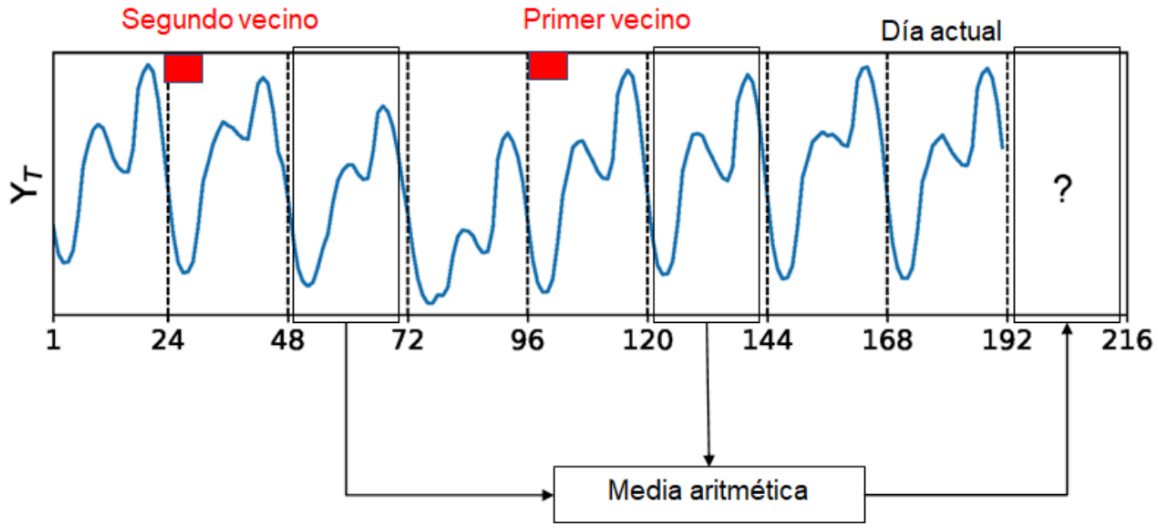
Este algoritmo necesita de varios datos de entrada para realizar con efectividad su función. Primero, se necesita el valor correspondiente al número de vecinos que vamos a tener en cuenta para realizar la operación. Después, requiere de un conjunto de datos a los cuales se le va a aplicar el algoritmo. El formato de estos archivos es el siguiente: la primera línea contiene dos números enteros separados por un espacio. Estos números indican cuántas filas hay en el archivo y cuántos datos hay por fila. A partir de la segunda línea, cada fila contiene la cantidad de datos especificada anteriormente. Los datos están separados por comas y utilizan el punto como separador decimal. Para finalizar, se debe incluir el número de hilos y de procesos que el programa va a requerir.

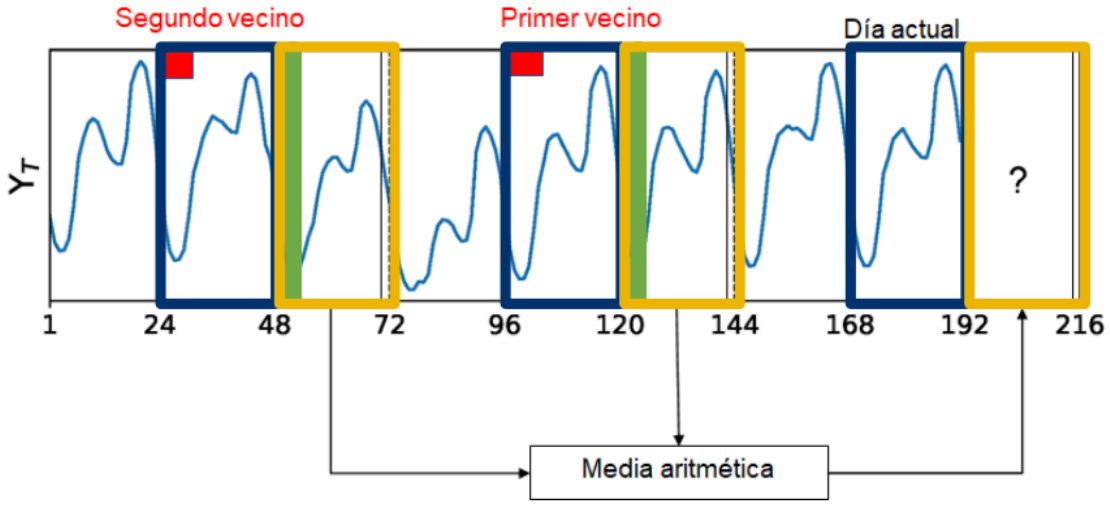
El algoritmo

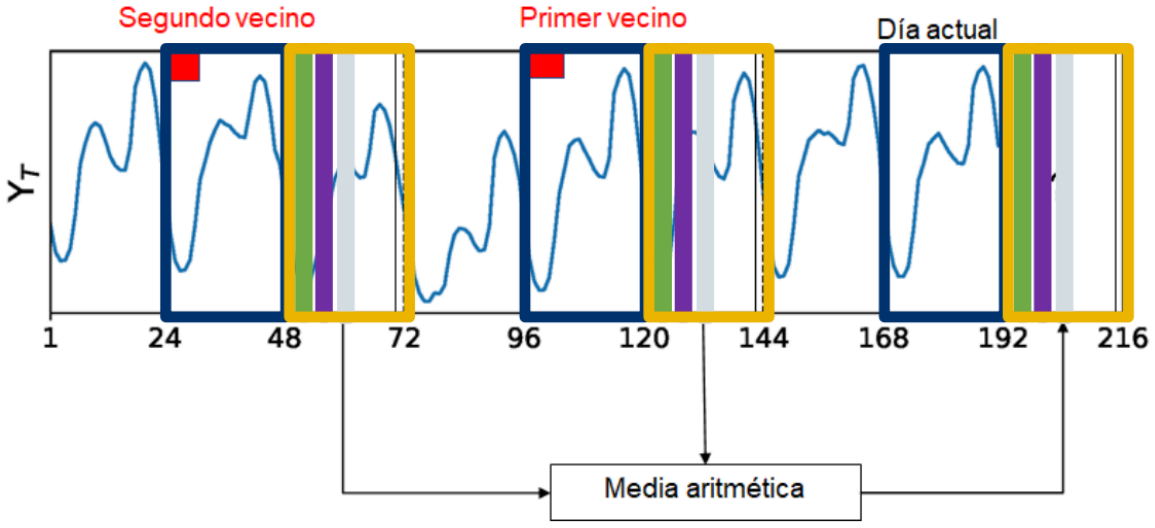
Para realizar una predicción, se comparará el día que se quiere predecir con otros anteriores con el objetivo de encontrar los días más parecidos. Siendo k un número establecido de días similares. De ese grupo de días se calculará la media por hora entre ellos para obtener una predicción. Esto es, en esencia, el algoritmo “k vecinos más cercanos”, del inglés, k-Nearest-Neighbors.

Este es un algoritmo considerablemente simple y apto para muchas posibles mejoras o modificaciones. Pese a su sencillez, puede ser utilizado para la resolución de numerosos problemas de distinta índole ya que su adaptación es relativamente fácil.

En este problema, resulta útil para realizar la predicción, pues su complejidad es apta para trabajar en un tiempo asequible.


Segundo vecino
Primer vecino
Día actual
Y_T
?
1
24
48
72
96
120
144
168
192
216
Media aritmética


Segundo vecino
Primer vecino
Día actual
Y_T
?
1
24
48
72
96
120
144
168
192
216
Media aritmética


Segundo vecino
Primer vecino
Día actual
Y_T
1
24
48
72
96
120
144
168
192
216
Media aritmética

Cómo se ha aplicado la solución

Debido a que las mediciones pueden llegar a alcanzar los cientos de miles de registros, resulta extremadamente lento realizar el análisis de tal cantidad de datos y operar con ellos. Para solucionar este inconveniente, es ideal hacer uso de la capacidad de paralelización con la que cada CPU cuenta con sus hilos y realizar una distribución entre el máximo de núcleos (procesos) posible, y así tratar de realizar la máxima cantidad de trabajo posible de manera simultánea.

Para distribuir la ejecución del código, se contará con varios ordenadores de varios núcleos y se interconectarán mediante mpich y clave pública. Será únicamente un proceso el encargado de enviar los datos y las instrucciones al resto, que pueden encontrarse tanto en la misma máquina como en otra conectada mediante la red.

Descripción de la implementación

Función init

En la función *init* comprobamos *argc* para comprobar que tenga los 3 parámetros que necesitamos. Usamos *filename* como parámetro de ruta al fichero de datos, *k* es el número de vecinos a buscar para calcular las medias en el algoritmo y *nt* es el número de hilos a usar en los casos de tareas distribuibles.

Con los argumentos del programa generamos, en dicha función, una estructura de tipo *knn_args* que contiene dichos datos para su posterior utilización.

Carga de ficheros

En la carga de ficheros hemos decidido hacerlo en el proceso 0 o máster, generando una matriz de *n* filas y *NHOURS* columnas siendo *n* el número de filas del archivo y *NHOURS* el número de mediciones que se hacen al día, en el caso actual 24.

Dicha carga de datos se realiza en la función *load_dataset* en la que comprobamos el PID y si es el correcto, es decir, es el máster llamamos a una función llamada *knn_load_dataset* que nos carga en una matriz *data* todos los datos del fichero correspondiente.

Broadcast de datos hacia los procesos

El proceso máster envía los datos necesarios para la búsqueda de vecinos a cada proceso esclavo. En primer lugar realiza un broadcast del número de días (número de columnas de nuestra matriz de datos) que se están manejando en el dataset actual. Una vez recibido el dato necesario referente al número de columnas del dataset se necesita repartir el conjunto de datos entre el número de procesos que vamos a usar, para ello calculamos los metadatos de los “chunks” que vamos a enviar a cada proceso, para ello

usamos la función *initialize_chunk_metadata*, que calcula los tamaños de los datos que se van a recibir en cada proceso.

En último lugar se distribuyen los datos con la función *MPI_Scatter* entre todos los procesos que se vayan a utilizar.

Encontrar vecinos

La función find_neighbors es una pieza fundamental del algoritmo k-NN, responsable de encontrar los k vecinos más cercanos para un individuo dado. Debido a la naturaleza intensiva de este cálculo, su eficiencia juega un papel crucial en el rendimiento general del algoritmo. Para optimizar su ejecución, se implementa una estrategia de paralelización que aprovecha el poder de cómputo de múltiples procesos.

Al dividir el conjunto de datos en varios chunks de información, la función find_neighbors distribuye estas partes entre los procesos disponibles. Cada proceso es responsable de calcular los vecinos más cercanos para su chunk asignado de manera simultánea

Hacer predicciones y calcular MAPE

Para hacer las predicciones se llama a la función knn predictions que se encuentra en el fichero knn.c. En esta función se accederá a los subgrupos de vecinos calculados previamente para realizar una media sumando los valores hora a hora y dividiendo por k. Con esto, el día a predecir queda establecido según la media de los vecinos seleccionados. Estas predicciones son guardadas en un buffer para posteriormente ser guardadas.

Dentro de esta misma función y en el mismo recorrido que en el necesario para calcular las predicciones, se va calculando el error cometido en la predicción.

El bucle que recorre dia a dia a predecir, está paralelizado de forma que sea repartido entre los hilos del proceso que lo está ejecutando.

Guardar predicciones

Una vez han sido calculadas las predicciones, tan solo queda guardarlas en el fichero Predictions.txt, que se ha abierto y comprobado previamente, recorriendo todos los registros a predecir y guardando fila a fila todas las horas calculadas.

Guardar MAPE

En la función knn save eMAPE se escriben los errores calculados previamente en el fichero MAPE.txt mediante un recorrido dia a dia.

Ejecución del algoritmo

Tiempos

El tiempo de ejecución es extremadamente importante en aquellos algoritmos cuyo objetivo ser ejecutados de forma distribuida en un cluster, pues es vital conocer la ubicación de cuellos de botella en el código fuente que puedan ralentizar la ejecución, ya sea por el excesivo trabajo de un proceso en especial o por una lenta distribución de los datos.

En casos en los que la cantidad de datos a analizar es poca, el tiempo no será un problema, pero cuando los ficheros de datos alcanzan el orden de GigaByte puede resultar altamente costoso para un algoritmo mal distribuido y paralelizado. Para mostrar el resultado se ha logrado con este algoritmo, a continuación se muestran unas gráficas del tiempo de ejecución en milisegundos con respecto a la cantidad de datos en MegaBytes.

Precisión

El manejo de la precisión es sumamente crucial para garantizar resultados confiables y precisos. Además, almacenar estos datos puede ayudarnos a mejorar la precisión en futuras predicciones y asegurar una mayor efectividad.

En nuestro caso, para calcular la precisión de los errores se ha utilizado la función MAPE (Mean Absolute Percentage Error). La función se expresa en porcentaje, por lo que es sencillo interpretar el resultado. Además, el MAPE es una medida relativa que no depende de la escala de los valores pronosticados o reales. Esto lo hace útil para comparar la precisión de diferentes modelos o métodos de pronóstico, independientemente de la magnitud de las variables. Todo esto sumado a la facilidad de cálculo y de implementación que posee hacen del MAPE una función ideal para este proyecto.

$$MAPE(\%) = \frac{100}{h} \sum_{n=1}^{h} \left(\frac{|Rn - Pn|}{|Rn|}\right)$$

Descripción Hardware

Ordenador Marcos V	
Procesador	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
Número de núcleos	8
Velocidad	4.2 GHz
Número de hilos	16
Memoria RAM	8.0 GB

Ordenador Antonio M	
Procesador	Intel® Core™ i7-9700F
Número de núcleos	8
Velocidad	4.7 GHz
Número de hilos	16
Memoria RAM	32.0 GB

Ordenador Antonio G	
Procesador	Intel® Core™ i7-9700F
Número de núcleos	8
Velocidad	4.2 GHz
Número de hilos	16
Memoria RAM	8.0 GB

Ordenador Pablo (Medidas de tiempo)	
Procesador	Intel® Core™ i7-9700F
Número de núcleos	8
Velocidad	4.7 GHz
Número de hilos	16
Memoria RAM	8.0 GB