Programación Concurrente

Trabajo Práctico

Se desea implementar en Java un programa que reconozca caracteres numéricos en imágenes de 28×28 píxeles representadas en escala de grises. Existe un conocido repositorio de 60000 imágenes con estas características, que ya vienen etiquetadas, llamado MNIST[1], creado en 1994. Podemos ver algunas imágenes de ejemplo:

8 4 0 3

La idea es reconocer nuevos dígitos por medio del algoritmo de *k-nearest neighbors* (kNN), en español, "los k vecinos más cercanos". Para entender cómo funciona esta estrategia, consideremos un ejemplo en 2 dimensiones, con esta imágen tomada de una referencia de IBM[2].

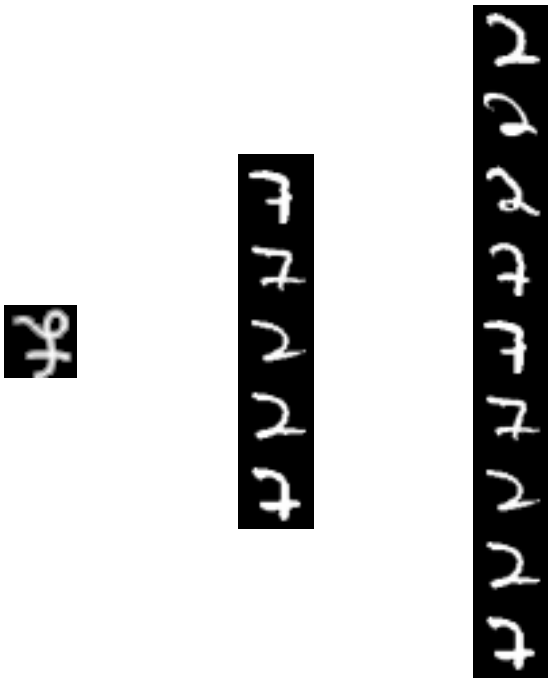
Consideremos un conjunto de puntos en $\mathbb{R}^2$ que tenemos clasificados en dos clases A y B (azul y verde en la imágen, respectivamente). En este caso, podemos apreciar que los puntos de cada clase se separan en dos cúmulos relativamente claros. Si se nos presenta un nuevo punto y tenemos que determinar si pertenece a la clase A o a la clase B, una posibilidad es buscar, por cercanía, a cuál cúmulo parece pertenecer. Para hacer esto, como se muestra en la figura del medio, la idea es considerar las clases de los puntos clasificados que tiene más cerca. En particular, lo que podemos hacer es, para algún parámetro k, buscar los k puntos clasificados más cercanos al punto nuevo, como muestra la figura de la derecha.

Ciertamente, las figuras de 28×28 del set MNIST no son puntos en $\mathbb{R}^2$. Sin embargo, podemos utilizar la misma idea, si pensamos en cada píxel de la imágen como un punto en un espacio de $28 * 28 = 784$ dimensiones. Cada imágen es entonces un vector que en cada coordenada tiene un valor entre 0 y 255, los valores posibles en escala de grises, y una clase que corresponde con el número representado entre los 10 del 0 al 9. Hay muchas maneras de calcular la distancia entre dos puntos $\mathbf{x}$ e $\mathbf{y}$ en el espacio, la más elemental es sumar los cuadrados de las diferencias entre sus coordenadas.

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{784} (\mathbf{x}_i - \mathbf{y}_i)^2$$

[1] `https://yann.lecun.com/exdb/mnist/`
[2] `https://www.ibm.com/topics/knn`

Luego, dado un punto en este espacio 784-dimensional (es decir, una imágen nueva), tenemos que calcular su distancia a cada punto de los 60000 conocidos y quedarnos con las clases de los k puntos más cercanos. Allí, la clase a la que pertenezca la mayoría será la que designemos como la respuesta[3]



En este ejemplo, utilizamos la figura de la izquierda. La columna del centro y la columna de la derecha corresponden a las 5 (resp. 9) figuras de MNIST más cercanas a ella según la distancia dist, donde más abajo significa más cercana. Por lo tanto, el algoritmo kNN con $k = 5$ determina que la figura es un "7", mientras que con $k = 9$ determina que es un "2".

Como la distancia con cada elemento del conjunto MNIST se puede calcular de manera independiente, la idea es que el cómputo de los k vecinos más cercanos se pueda hacer de manera concurrente entre múltiples threads.

Datos MNIST de *entrenamiento* y de *prueba*

Para este trabajo práctico vamos a usar los datos de MNIST pasados a formato `csv` por Joseph Redmon [4], que van a estar subidos en el drive con el material de la materia en el archivo `mnist.zip`. Hay dos archivos, `mnist_train.csv` que corresponde a las 60000 imágenes que usaremos en nuestro repositorio, y `mnist_test.csv`, que son 10000 imágenes adicionales que también están etiquetadas pero que usaremos para poner a prueba nuestro identificador.

Los dos archivos `csv` tienen el mismo formato: cada línea corresponde con una imágen, y son 785 valores separados por comas. El primer valor corresponde a la etiqueta, es decir, el número representado, y las 784 restantes corresponden con el valor de gris ($0 = negro$, $255 = blanco$) de un píxel de la imágen, por fila (los primeros 28 son la primera fila, los segundos 28 la segunda, ..., etc.). En el apéndice se muestra un ejemplo

[3] Notar que puede haber empate. Existen distintas estrategias para desempatar, para este trabajo vale usar la que prefieran, o, mejor, probar varias y encontrar la que sea más efectiva.

[4] `https://pjreddie.com/projects/mnist-in-csv/`

de cómo convertir estos valores a una imágen de necesitarlo (así como también como llevar una imágen a este formato).

Objetivo del Trabajo Práctico

Se pide implementar un programa en Java con dos funcionalidades: Por un lado, debe ser capaz de, dada una figura de 28×28 en escala de grises y formato `png`, determinar a qué número corresponde según el algoritmo kNN. El valor de k debe ser configurable, así como la cantidad de *threads* a usar. Asimismo, el programa debe tener un "modo test", capaz de tomar como entrada un archivo `.csv` con el formato de `mnist_train.csv` y `mnist_test.csv`, línea por línea determinar por kNN el número correspondiente a esa línea, y compararlo con la etiqueta en el archivo. Al terminar, se debe imprimir una medida de *precisión*, es decir, la proporción de veces que la predicción de kNN coincidió con la etiqueta efectiva (Si el programa funciona correctamente, es esperable una precisión superior a 0,9).

Se pedirá tomar los tiempos que lleva evaluar una imágen y el que lleva testear un `csv` con distintas cantidades de casos, configurando también la cantidad de threads utilizados, de manera de determinar el impacto en la performance de aumentar la cantidad de threads trabajando.

Elementos a entregar

Programa Java

Se debe presentar un programa escrito en Java que permita modificar los siguientes parámetros: `modo` (imagen simple o archivo de test), `k`, `cant_threads`, y `tamano_buffer`. El programa deberá tener la siguiente estructura:

1. Una clase `Main` con el punto de entrada del programa, que tiene la responsabilidad de inicializar las estructuras mencionadas a continuación y producir unidades de trabajo consistentes en subconjuntos del set de datos de entrenamiento para que sus distancias sean calculadas por múltiples thredads. El programa debe terminar únicamente cuando haya concluído el cómputo (ya sea declarando cuál es el número de una imágen o determinado la precisión para un archivo `.csv`), y debe informar el tiempo tomado por el cómputo.

2. Una clase `Buffer` (implementada como un monitor utilizando métodos synchronized) que actúa como una cola FIFO concurrente de capacidad acotada. Es decir, bloquea a un lector intentando sacar un elemento cuando está vacía y bloquea a un productor intentando agregar un elemento cuando está llena. La capacidad del Buffer es la que corresponde al parámetro `tamano_buffer`.

3. Una clase `WorkerCounter` (implementada como un monitor utilizando métodos synchronized) que evita que `Main` termine su ejecución mientras queden threads trabajando.

4. Una clase `Worker` que extienda de `Thread` y que tome tareas (clase `Task`) de un buffer hasta tomar una señal de terminación (*Poison Pill*).

5. Una clase `Task` que implementa `Runnable`, con una subclase `MNISTask` que realiza las comparaciones contra un subconjunto de los datos de entrenamiento. El método `run` de cada `MNISTask` toma las filas del conjunto y evalúa su distancia contra un determinado punto. Se recomienda usar conjuntos que sean lineas consecutivas

del `csv` original (por ejemplo, las primeras 10000, luego las segundas 10000, etc. hasta cubrir las 60000 filas). Notar que va a ser necesario juntar los resultados de todas las tasks para conocer cuáles son efectivamente los k puntos más cercanos. Esto puede lograrse por medio de un monitor que vaya acumulando los resultados parciales o con una `Task` distinta a `MNISTask`.

6. Una clase `PoisonPill` que hereda de `Task` y que hace que el `Worker` que la tome termine su ejecución (puede ser por medio de una excepción, como en la práctica).

7. Una clase `ThreadPool`, que se encarga de instanciar e iniciar la cantidad de `Workers` pedida por un usuario.

8. Cualquier otra clase que considere necesaria (Por ejemplo, la que necesite para obtener los k elementos más cercanos, o para contabilizar los aciertos).

Al iniciar el programa la clase `Main` debe delegar la iniciación de los threads necesarios en la clase `ThreadPool` y luego introducir de a uno los conjuntos a procesar en el `Buffer`. Cada `Worker` en funcionamiento debe tomar las tareas de a una del `Buffer` y hacer las comparaciones que le correspondan. Cabe destacar que es inadmisible utilizar una cantidad de threads menor a la solicitada por el usuario.

El programa, además, debe estar correctamente documentado. En particular, debe contener las instrucciones para configurar los parámetros, y el nombre del archivo de salida. Idealmente hacer que el programa se pueda ejecutar por línea de comandos y tome estas opciones por parámetro o por entrada estándar. No es incorrecto que el programa haga impresiones por pantalla intermedias para informar su progreso, pero se pide evitar la impresión de mensajes de *debug* (puede existir una opción para ponerlos pero por defecto debe estar apagada).

Informe

Además del código, se requiere un informe corto en formato `pdf`, donde se deben mostrar los resultados de la aplicación del programa sobre alguna entrada dibujada por el grupo. Se debe determinar qué número se asigna al dibujo cuando $k = 1$, $k = 5$ y $k = 10$. Asimismo, se debe registrar el tiempo que tarda el programa en determinar el número del dibujo cuando se usa sólo 1 thread y cuando se usan 4 threads.

A partir de `mnist_test.csv` se deben crear archivos de prueba de distintas cantidades de líneas para evaluar la precisión y tiempo en procesarlos. Registrar el tiempo de ejecución del programa utilizando distintas combinaciones de cantidades de threads y cantidades de líneas.

- **Threads:** 1, 4, 8 y 16
- **Cant. de líneas en `csv` de pruebas:** 500, 1000, 2000, 5000

El informe, que es **condición necesaria para la aprobación del TP**, debe respetar el siguiente formato:

1. **Autoría:** Nombres, e-mail y número de legajo de quienes hicieron el trabajo (máximo 3 personas salvo excepciones acordadas con los docentes).

2. **Introducción:** Sección explicando el dominio del TP, el diseño del código y cualquier consideración adicional que considere relevante para la correcta interpretación de los resultados.

3. **Evaluación:** Sección explicando el proceso de evaluación, incluyendo los detalles del hardware donde se ejecutan las pruebas (modelo de microprocesador con cantidad de núcleos, memoria disponible y versión del sistema operativo). En esta sección deben incluirse los tiempos demorados para el procesamiento de la imágen con 1 y 4 threads, además de discutir las distintas respuestas al variar el k (si las hubo). Opcionalmente, poner los dibujos de las k imágenes en el set de entrenamiento que más se parecieron a la imágen nueva. En cuanto a las pruebas con el `csv`, poner la tabla con las precisiones y tiempos de ejecución para cada combinación de cantidad de threads y cantidad de líneas. Agregar un gráfico con 4 líneas, una por cada cantidad de threads, que tenga en el eje x la cantidad de líneas del set de pruebas y en el eje y el tiempo de ejecución utilizado. Si para una o varias de estas combinaciones el tiempo de ejecución es muy pequeño y produce mucha variabilidad, tomar un promedio de 10 ejecuciones. Esto debe estar documentado.

4. **Análisis:** Sección en la que se debe discutir los datos obtenidos en la evaluación, en particular destacando la combinación de parámetros con la que se obtuvo el tiempo de ejecución mínimo y la combinación que obtuvo el tiempo máximo, más alguna conclusión (Por ejemplo, responder a ¿Es verdad que aumentar la cantidad de threads siempre mejora el rendimiento? ¿Influye en algo variar el k? ¿Qué prueba adicional podría realizarse?)

Forma de Entrega

Se deben enviar el archivo `.zip` con el código del programa y el `.pdf` con el informe por email a las casillas de correo electrónico de *ambos* profesores con el subject:
`Entrega TP PCONC 2S2024 - (apellido1) - (apellido2) - (apellido3)`
(donde `apellido1`, `apellido2` y `apellido3` son los apellidos de las personas que constituyen el grupo). El mensaje debe ir *con copia* a las otras personas que integre el grupo, pues la devolución del TP se hará en *respuesta a todos* a ese correo.

Es posible trabajar en un repositorio *git* compartido por las personas que compongan el grupo. En este caso, el repositorio debe ser **privado**. Al entregar, se deberá agregar a los profesores al repositorio. De todos modos se solicita que envíen el mail con las condiciones especificadas, aunque en vez de tener los archivos adjuntos incluirán el link al respositorio. Se aclara que se espera que el código entregado por el grupo sea **original**. Si se detecta que el mismo fue copiado de alguna fuente online será automáticamente desaprobado sin posibilidad de reentrega.

Fecha de Entrega

El TP debe entregarse antes del **Martes 19 de Noviembre** a las **23:59hs**. En caso de ser necesario, se solicitará una reentrega antes del **Martes 10 de Diciembre**.

Apéndice: Algunas herramientas de Java

Lectura de archivos `.csv`

Se puede hacer con las clases `FileReader` y `BufferedReader`, aplicando `split()` a cada línea obtenida.

```
try (BufferedReader br = new BufferedReader(new FileReader("archivo.csv"))) {
  String line;
  for (int i=0;i<lineas;i++) {
        line = br.readLine();
```

```
        String[] vals_str = line.split(",");
    //En el array vals_str estan los valores, en tipo String
  }
}
```

Extracción de valores de una imágen

Para leer una imágen y transformarla en valores como los del archivo `csv`, podemos usar `BufferedImage`, de la cual se puede extraer el `DatabufferByte`, que es un array de `byte`.

```
File imgPath = new File("cifra.png");
BufferedImage bufferedImage = ImageIO.read(imgPath);
WritableRaster raster = bufferedImage.getRaster();
byte[] datos   = ((DataBufferByte) raster.getDataBuffer()).getData();
```

Asimismo, dado un array de `byte` del tamaño adecuado (por ejemplo $28 \times 28 = 784$), se puede generar una imágen con:

```
byte[] datos;

BufferedImage imagen = new BufferedImage(28, 28, BufferedImage.TYPE_BYTE_GRAY);
imagen.getRaster().setDataElements(0, 0, 28, 28, datos);
File outputfile = new File (nombre_archivo);
try {
 ImageIO.write(imagen, ‘‘png’’, outputfile);
} catch (IOException e) {
}
```

Casteos entre `String`, `int` y `byte`

Será necesario hacer algunos casteos de tipos de datos para poder hacer todas las operaciones necesarias. En particular, es posible pasar de `String` a `int` con `Integer.parseInt()`, mientras que es posible pasar de `byte` a `int` con `Byte.toUnsignedInt()` (Es importante que sea **unsigned**). Si se quiere hacer el casteo de `int` a `byte`, basta con hacer el casteo directo `(byte)`.

Obtener las k distancias mínimas

Si bien no es la única forma, una estructura de datos que puede ser útil para obtener las k distancias mínimas es el `PriorityQueue`. `PriorityQueue` es un tipo parametrizado, se puede crear uno con cualquier clase que implemente `Comparable`, y los elementos se insertan ordenados de menos a mayor. Dada una `PriorityQueue`, se le puede hacer `add()` para agregar un elemento, y `poll()` para obtener y quitar el mínimo.

Luego, si en una `PriorityQueue` se ponen todas las distancias y se hace `poll()` k veces, se tienen las k distancias mínimas (de todos modos, posiblemente haya que guardar una clase más compleja que sólo las distancias).

Observar que `PriorityQueue` **no es thread-safe**. Eso quiere decir, si múltiples threads quieren hacer operaciones sobre una misma instancia de `PriorityQueue`, no hay garantía de que esta funcione bien. Deberán implementar algún mecanismo para sortear esto si utilizan esta estructura de datos.

Medición de tiempos

Para medir el tiempo de ejecución del programa, se puede llamar al método provisto por Java `System.currentTimeMillis()`. Se debe guardar el tiempo actual cuando se va a comenzar a operar (después de cargar los datos de entrenamiento), y cuando se terminó la ejecución (es decir, luego de que los Workers hayan parado). Luego, basta con hacer la diferencia entre ambos tiempos y dividirla por mil para obtener el tiempo demorado expresado en segundos.