Reporte tarea 4 - Programación y algoritmos

Erick Salvador Alvarez Valencia

Centro de Investigación en Matemáticas, A.C.,
erick.alvarez@cimat.mx

Resumen En el presente reporte se analizará la elaboración de un analizador de texto, el cual toma como entrada los nombres de 10 libros, escritos por dos autores distintos, los divide en clases, toma aleatoriamente muestras de ambos conjuntos y genera una clasificación de los mismos, la cual se explicará más detalladamente en la descripción del programa. Una vez la clasificación termina, el programa genera un archivo con los resultados. A continuación se hablará sobre cómo trabaja el clasificador, de la misma forma se brindará un poco de pseudocódigo y al final del reporte se hará el análisis de complejidad del analizador.

Keywords: Lenguaje C, clasificador, texto.

1. Descripción

El presente programa genera una clasificación sobre dos conjuntos de textos, dichos conjuntos son los libros de la serie "Mago de Oz", los cuales han sido escritos por diversos autores, para este caso solo tomamos diez libros de 2 autores.
Antes que nada hay que mencionar que se han usado las librerías generadas en las tareas 2 y 3, las cuales son: las funciones de alicación y liberación de memoria, el separador de archivos y el analizador de frecuencias. Todas estas librerías complementan en gran medida al presente programa, aunque hay que decir que tuvieron que ser modificadas para trabajar en conjunto con el código de este programa.
En general las modificaciones son pequeñas ya que se quiere preservar la funcionalidad general del programa. Para las funciones de memoria se crearon varias que trabajan con matrices y vectores de diferentes tipos en lugar de un genérico, y para el separador de texto y analizador de frecuencias se adaptaron para automatizar el procesamiento de varios archivos.

El analizador de texto depende de que ya exista la tabla de frecuencias relativas, la tabla es de la forma (número de bloques) * (número de palabras), la tabla debe contener todos los bloques de los dos conjuntos de datos, una vez se tiene la tabla, se generan dos matrices en base a la misma, las cuales son de los bloques de los libros del primer autor y del segundo. Al tener dichas matrices, generamos un arreglo de enteros por cada matriz, el cual denota cuál será el conjunto de entrenamiento y el de prueba, dicho arreglo se genera aleatoriamente. Ahora que se sabe cuáles son los conjuntos de entrenamiento y cuáles de prueba se procede a realizar la clasificación, la cual toma cada vector de palabras denotando al bloque_i y lo compara con todos los vectores de palabras que están en las clases 1 y 2 y que son de entrenamiento. La comparación de los vectores de palabras se realiza usando la distancia euclideana y con un vector binario generado aleatoriamente. A continuación se muestra la fórmula de la distancia euclideana:

$$\sum_{i}^{n} (\vec{a}_i - \vec{b}_i)^2 * \overrightarrow{bin}_i$$

A continuación se despliega el pseudocódigo de la función de clasificación.

Algorithm 1 Clasificador de textos.

```
1: procedure CLASIFY(A, B, n, m)
2:     minV ← INF.
3:     clasei ← Clase que tendrá el elemento A_i.
4:     for i ← 1ton do
5:         if A_i es del conjunto de prueba then
6:             for j ← 1tom do
7:                 if B_i es del conjunto de entrenamiento then
8:                     aux ← dist(A_i, B_j)
9:                     if aux < minV then
10:                        minV ← aux
11:                        clasei ← Clase(B_i)
12:    return clasei
```

Donde A y B son los conjuntos de datos de la clase 1 y clase 2, y n y m son sus tamaños.

Para lo anterior requerimos tener bien en claro quiénes son nuestros conjuntos de prueba y entrenamiento, para lo cual requerimos de una función separadora que realice ese trabajo. Una idea para hacer esto anterior es tener definidas las matrices de clase 1 y clase 2 y generar dos vectores, los cuales dirán si el conjunto de palabras del bloque con índice i es de entrenamiento o de prueba. El algoritmo para hacer esto anterior es el siguiente:

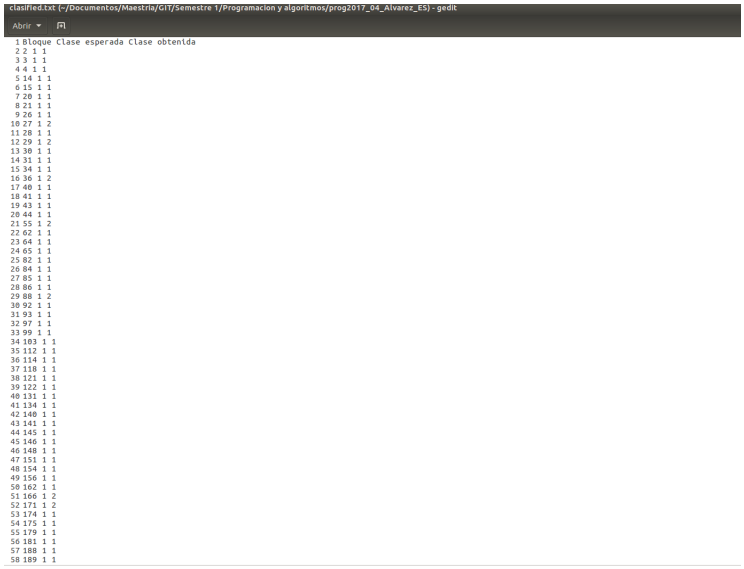
Algorithm 2 Generador de datos de entrenamiento y prueba.

```
1: procedure GENERATEDATA(A, n)
2:     arr ← Arreglo de tamaño n.
3:     sz ← round(n * 0,3).
4:     cnt ←= 0.
5:     while cnt < sz do
6:         rn ← rand(0, n − 1).
7:         if arr[rn] = 0 then
8:             arr[rn] ← 1.
9:             cnt ← cnt + 1.
10:    return arr
```

De esta manera tenemos un vector el cual nos indica cuáles índices son del conjunto de entrenamiento y cuales son de prueba, respetando las distribuciones $70 - 30$.

2. Descripción de las ejecuciones

Se realizaron varias ejecuciones con diferentes semillas aleatorias, esto para ver la calidad de la clasificación con distintos conjuntos de prueba y entrenamiento, así como con diferentes vectores binarios. Se pudo observar que la calidad de las clasificaciones es bastante buena, en promedio habían de 8 a 9 valores que no coincidian con su clase determinada.
A continuación se mostrará una captura de pantalla al resultado de una ejecución hecha con los diez libros.



(a) Figura 1. Resultado de la ejecución del programa con una semilla aleatoria.

Pese a que no se aprecia la lista completa de valores, podemos observar que en los bloques 27, 29, 36, 55, 88, 166, 171 fueron clasificados en las clases incorrectas. El error general de la clasificación es de aproximadamente siete por ciento.

3. Conclusiones del programa

Queda concluir el programa funcionó según lo esperado, hizo una clasificación donde la mayoría de los valores fueron correctamente clasificados. De la misma forma el programa se corrió usando la herramienta de valgrind y la misma demostró que toda la memoria fue correctamente liberada.
Podemos observar que existen varias mejoras que se pueden hacer al algoritmo. Primeramente la creación de 2 submatrices en base a una general es innecesario ya que podemos trabajar directamente con esta misma. Ahora la parte de la clasificación tiene un costo computacional alto ya que se deben de hacer varios barridos para todos los datos de prueba. Se cree que hay algoritmos más eficientes que puedan realizar la misma o una mejor clasificación con un menor costo computacional, como los que usan técnicas de machine learning, modelos bayesianos, n-gramas, etc.

4. Parte 2

Definimos la distancia de un punto p a un conjunto de puntos A, como la mínima distancia euclidiana entre p y un elemento de A.
Definimos la distancia entre dos conjuntos A y B como la mínima distancia entre el conjunto en A y un punto en B. Si los puntos están en R^d, y A tiene n y B tiene m.

1. **Calcular la distancia de un punto a A.** Definimos a A como un conjunto de puntos y *dist* como una función que calcula la distancia euclideana entre dos puntos. Entonces tenemos:

Algorithm 3 Distancia mínima entre un punto y un conjunto A.

```
1: procedure MinDist(A, p, n)
2:     minV ← INF.
3:     for i ← 1ton do
4:         aux ← dist(A_i, p)
5:         if aux < minV then
6:             minV ← aux
7:     return minV
```

Para este algoritmo podemos observar que las asignaciones, el if y la llamada a la función *dist* corren en tiempo constante $O(1)$, en cambio el ciclo for de la línea 3 depende del tamaño de n. Por lo que podemos definir cuatro constantes C_i tal que $T(n) = C_1 + n * C_2 * (C_3 + C_4)$ donde n es el tamaño del conjunto A. Es fácil darse cuenta que la función $T(n)$ es lineal y acotada por una recta, por lo que su complejidad es $O(n)$.

2. **Calcular la distancia entre los conjuntos A y B.** Definimos A, B y *dist* de la misma forma que en el caso anterior. Por lo que tenemos:

Algorithm 4 Distancia mínima entre un conjunto A y un conjunto B.

```
1: procedure MinDistSets(A, B, n, m)
2:     minV ← INF.
3:     for i ← 1ton do
4:         for j ← 1tom do
5:             aux ← dist(A_i, B_j)
6:             if aux < minV then
7:                 minV ← aux
8:     return minV
```

Para este algoritmo podemos observar que las asignaciones, el if y la llamada a la función *dist* corren en tiempo constante $O(1)$, en cambio el ciclo for de la línea 3 depende del tamaño de n y el de la línea 4 depende del tamaño de m. Otra cosa que podemos observar es que por cada ejecución del primer for, el segundo se ejecuta completamente, esto es, las instrucciones dentro del for de la variable j se ejecutan m veces por cada ejecución del primer for. Así que tenemos 5 constantes C_i tal que: $T(n, m) = C_1 + n * C_2 * (m * C_3 * (C_4 + C5))$.

En el análisis anterior nos podemos dar cuenta que la función T depende de n y m y que, gracias a la composición de los for la función es de grado dos, una forma más fácil de ver esto es pensando en el peor caso posible, donde n y m tomen los mayores valores posibles, sin pérdida de generalidad podemos suponer que dichos valores son iguales, por lo que tenemos $n = m$, así que la función de es orden $O(n * n) = O(n^2)$.

3. **Clasificar los puntos en B (conjunto de entrenamiento) utilizando el vecino mas cercano en A (conjunto de prueba).** Para este punto se entiende que hay que realizar la parte de clasificación, tomando en cuenta el conjunto de entrenamiento como el 70 % y el de prueba como el 30 %. Por lo que el algoritmo quedaría como:

Algorithm 5 Clasificador con el vecino más cercano.

```
1: procedure MinDistSets(A, B, n, m)
2:     minV ← INF.
3:     for i ← 1ton do
4:         if A_i es del conjunto de prueba then
5:             for j ← 1tom do
6:                 if B_i es del conjunto de entrenamiento then
7:                     aux ← dist(A_i, B_j)
8:                     if aux < minV then
9:                         minV ← aux
10:    return Clase(minV)
```

Este algoritmo es muy parecido al del caso anterior, la diferencia es que sólo el 30 % de los datos del conjunto A entran al for interno, y sólo el 70 % de los datos del conjunto B se ejecutan de dicho for. Por lo que a resumidas cuenas la complejidad del algoritmo la definimos como: $T(n) = C_1 * 0{,}3 * n * (C_2 * 0{,}7 * m)$ aunque hay que recordar que en un análisis de big-O riguroso podemos despreciar las constantes ya que el polinomio queda acotado por su mayor grado, así que tenemos $O(n*m) = O(n^2)$ usando el mismo análisis del caso previo.