

Práctica 04

DOCENTE	CARRERA	CURSO
Vicente Machaca Arceda	Maestría en Ciencia de la Computación	Estructura de Datos y Algoritmos

PRÁCTICA	TEMA	DURACIÓN
04	Kd-Tree	3 horas

1. Datos de los estudiantes

- Grupo: 9
- Integrantes:
 - Abarca Murillo, Jhonatan Piero
 - Apari Pinto, Christian Timoteo
 - Suca Velando, Christian Anthony
 - Vargas Zuni, Arturo
- Repositorio: https://github.com/jabarcamu/EDA_Practica4
- Video: <https://youtu.be/2EECiQbAocA>

2. Ejercicios

2.1. Cree un archivo *main.html*

```

1 <head>
2   <title>Kdtree</title>
3   <script src="lib/bootstrap.min.js"></script>
4   <script src="lib/p5.min.js"></script>
5   <script src="js/kdtree.js"></script>
6   <script src="js/sketch.js"></script>
7
8   <link rel="stylesheet" href="css/bootstrap.min.css">
9
10 </head>

```

Listing 1: index.html

La distribucion de los archivos se mantienen por carpetas, en la carpeta js se tienen los archivos javascript ejecutables , tanto del arbol kdtree como del graficador usando la libreria p5.js, en la carpeta css los estilos que manejará, en este caso se optó por la libreria bootstrap.

2.2. Cree un archivo *kdtree.js*

A medida que va aumentado la dimensión del espacio d , como por ejemplo el número de atributos, cada nivel de descomposicion del quadtree resulta en muchas nuevas celdas o cuadrantes mientras vamos aumentando la dimension llegando a ser muy alta, por ejemplo 2^n . Esta situacion es mitigada

haciendo uso de variantes de un $k - dtree$, en el que k demota la dimensionalidad de el espacio a ser representado, si embargo en muchas aplicaciones la conversion llega a ser que el valor de d denota la dimension, y esto es en la practica el camino que elegimos seguir. En principio $kd - tree$ es un arbol binario donde el espacio subyacente es particionado en la base del valor de solo un atributo en cada nivel del árbol en lugar de la base de los valores d como en el caso de Quadtree. En otras palabras, la distinción con Quadtree es que en $kd - Tree$ solo un atributo, o la clave, es evaluado cuando determina la dirección en que cada rama sera hecha [1].

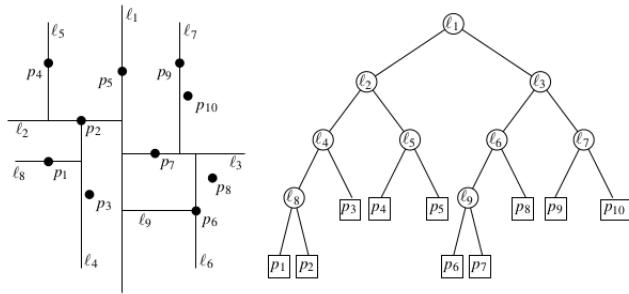


Figura 1: kd-Tree [2]

Originalmente el nombre se entiende como un arbol de k -dimensiones; tomando un ejemplo, el generar un arbol de 2-dimensiones será llamado como un $2d - tree$ [2], originalmente este término a sido olvidado actualmente solamente se conoce como 2-dimensiones kd-tree como se muestra en la figura 1.

Para la construcción del $kd - tree$ sera a partir de las siguientes características mostradas en el siguiente código. Un atributo punto (point) que almacena el punto en la k-dimensión, puntero a la izquierda (left) apuntador a la generacion del hijo izquierdo, puntero a la derecha y finalmente el eje al que actualmente se estara evaluando, en la parte de la construcción del kd-tree se explica con mayor detalle.

```

1 class Node {
2   constructor(point, axis) {
3     this.point = point;
4     this.left = null;
5     this.right = null;
6     this.axis = axis;
7   }
8 }
```

Listing 2: kdtree.js

2.3. Complete las funciones

2.3.1. build_kdtree: Construye el KD-Tree y retorna el nodo raiz.

Podemos construir un $kd - tree$ con un procedimiento recursivo que se describe a continuación. Este procedimiento tiene dos parámetros: un conjunto de puntos y un número entero. El primer parámetro es el conjunto para el que queremos construir el $kd - tree$, inicialmente esta es el conjunto P. El segundo parámetro es la profundidad de recursividad o, en otras palabras, la profundidad de la raíz del subárbol que construye la llamada recursiva. El parámetro de profundidad es cero en la primera llamada. La profundidad es importante porque determina si debemos partir con una línea vertical u horizontal. El procedimiento devuelve la raíz del $kd - tree$ [2].

```

1 function build_kdtree(points, depth = 0) {
2   // verifica si esta vacio
```

```

3     var n = points.length;
4     var axis = depth % k;
5
6     if ( n<=0){
7         return null;
8     }
9     if(n==1){
10        return new Node(points[0],axis)
11    }
12    var median = Math.floor(points.length/2);
13
14    points.sort(function(a,b){
15        return a[axis] - b[axis];
16    });
17
18    var left = points.slice(0,median);
19    var right = points.slice(median+1);
20
21    var node = new Node(points[median].slice(0,k),axis);
22    node.left = build_kdtree(left,depth +1);
23    node.right = build_kdtree(right,depth +1);
24
25    return node;
26 }
```

Listing 3: buildKdtree

2.3.2. getHeight: Retorna la altura del árbol

Verifica si el árbol no esta vacio, luego recursivamente retornamos la altura del nodo izquierdo y derecho, respecto al valor que estamos por recibir tanto de la altura izquierda y derecha comparamos si dicha altura es mayor en una de las ramas, así tomar como altura del árbol al lado mayor según la comparación realizada

```

1 function getHeight(node) {
2     if (node == null) {
3         return 0;
4     }
5     // acumulador de alturas por hijo
6     // var height_left = 0
7     // height_left += node.left +1;
8
9     var height_left = getHeight(node.left) + 1;
10    var height_right = getHeight(node.right) + 1;
11
12    // comparar si el acumulador es mayor en el hijo izq o derecho
13    // para dar la altura completa del arbol
14    if (height_left > height_right) {
15        return height_left;
16    }
17    else {
18        return height_right;
19    }
20 }
```

Listing 4: getHeight

2.3.3. generate dot: Genera al árbol en formato dot

Este procedimiento es similar al encontrar la altura del árbol, con la diferencia que solamente requerimos solamente imprimir el punto padre que este apuntando a un hijo recursivamente para luego obtener el resultado mediante el graficador Graphviz de Python.

```

1  function generate_dot(node) {
2      if (node === null) {
3          return "";
4      }
5
6      var tmp = '';
7
8      if (node.left != null) {
9          tmp += ' ' + node.point.toString() + ' ' + ' -> ' + ' ' + node.left.point.
10         toString() + ' ' + ';' + '\n';
11         tmp += generate_dot(node.left);
12     }
13     if (node.right != null) {
14         tmp += ' ' + node.point.toString() + ' ' + ' -> ' + ' ' + node.right.point.
15         toString() + ' ' + ';' + '\n';
16         tmp += generate_dot(node.right);
17     }
18
19     return tmp;
20 }
```

Listing 5: generateDot

2.4. Cree un archivo *sketch.js* y evalúe sus resultados

Utilizando la librería gráfica p5.js, iniciaremos el dibujo general de los puntos 800x800 para que la generación se tenga mayor visibilidad de los puntos generados en adelante, en este caso solo se uso la dimensión de 500x500, y para la generación de los puntos se utilizo la variable *particleCount* para generar los puntos aleatorios, para este caso ejemplo se tomaran 100 puntos aleatorios, finalmente la generación de cada punto esta dimensionada por una grilla interna para distinguir los sectores de separación de los puntos como se muestra en la figura 2

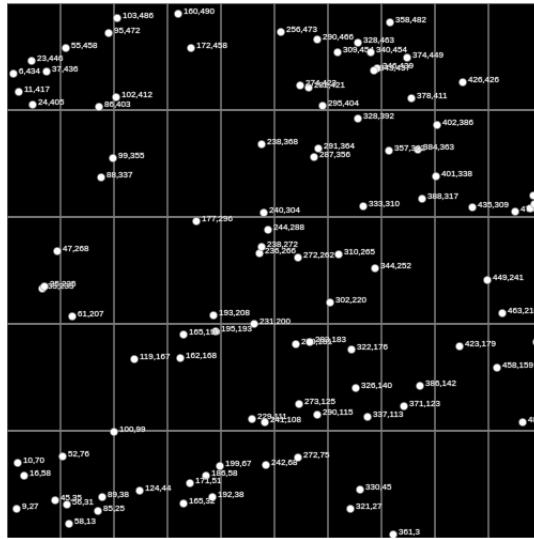


Figura 2: sketch.js, dibujando los puntos

Respecto a los puntos generados aleatoriamente, para el caso mostrado en el siguiente código, la dimensión de los puntos es de 2-dimensiones y iniciamos el árbol llamando a la función *buildKdtree* pasando como parámetros a los puntos generados, ver figura 3.

```
  rNode
    axis: 0
  ▾ left: Node
    axis: 1
  ▾ left: Node {point: Array(3), left: Node, right: Node, axis: 0}
  ▾ point: (3) [193, 208, 129.36]
  ▾ right: Node {point: Array(2), left: Node, right: Node, axis: 0}
  ▾ _proto__ : Object
  ▾ point: (3) [270, 181, 158.69]
  ▾ right: Node
    axis: 1
  ▾ left: Node {point: Array(2), left: Node, right: Node, axis: 0}
  ▾ point: (2) [435, 309]
  ▾ right: Node {point: Array(2), left: Node, right: Node, axis: 0}
  ▾ _proto__ : Object
  ▾ point: (2) [111, 111]
```

Figura 3: sketch.js, iniciando el *kd-tree*

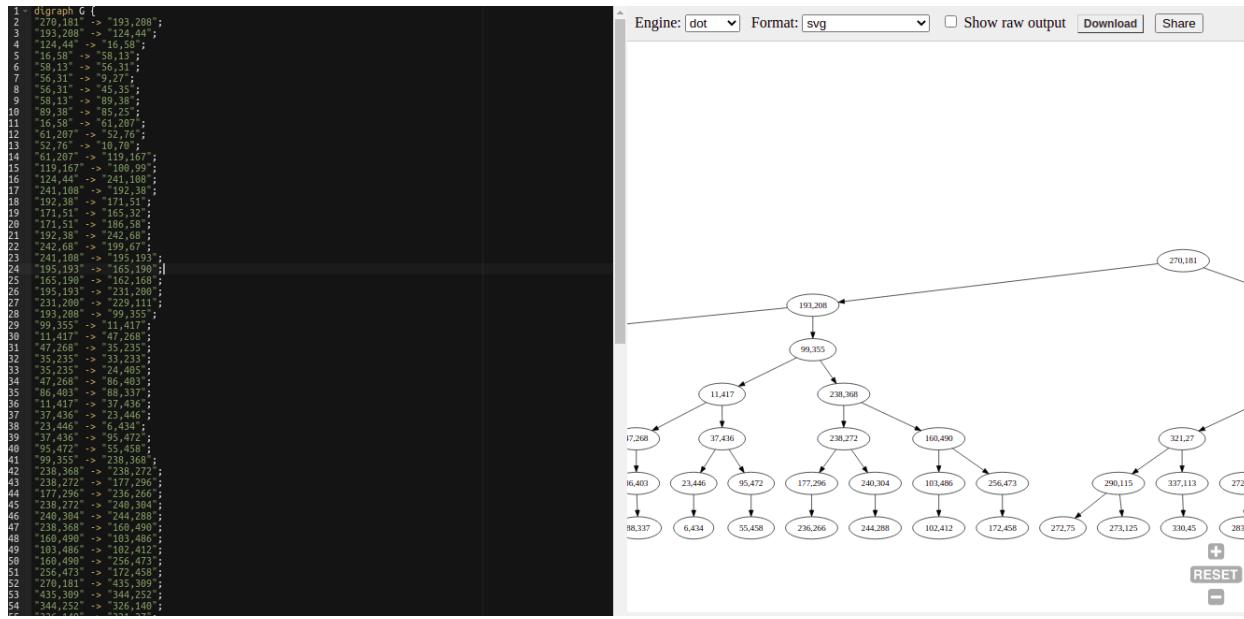
```
1 function setup() {
2     var width = 800;
3     var height = 800;
4
5     let dibujo = createCanvas(width, height);
6     dibujo.parent("dibujo");
7
8     background(0);
9     for (var x = 0; x < width; x += width / 10) {
10        for (var y = 0; y < height; y += height / 5) {
11            stroke(125, 125, 125);
12            strokeWeight(0.5);
13            line(x, 0, x, height);
14            line(0, y, width, y);
15        }
16    }
17
18    for (let i = 0; i < particleCount; i++) {
19        var x = Math.floor(Math.random() * height);
20        var y = Math.floor(Math.random() * height);
21        data.push([x, y]);
22
23        fill(255, 255, 255);
24        circle(x, height - y, 7); //200-y para q se dibuje apropiadamente
25        textSize(8);
26        text(x + ',' + y, x + 5, height - y); //200-y para q se dibuje apropiadamente
27    }
28
29    root = build kdtree(data);
```

Listing 6: sketch.js

Así mismo ya creado el *kd-tree* empezaría a llamarse a las otras funciones iniciales como la función *getHeight* que devuelve

Altura del Arbol es 7

Finalmente el resultado de los puntos desde generateDot para graficarlos en GraphViz, como se ve en la figura 4


 Figura 4: sketch.js, resultado de *generate_dot*

2.5. Implemente la función closest point brute force y naive closest point

Previamente para generar los dos siguientes algoritmos se requiere de la creación de la función de distancia Euclidiana o cuadrática, se define la siguiente función que recibe como parámetros dos puntos de la misma dimensión y calcula su distancia en ellas como se muestra en el siguiente código.

```

1 function distanceSquared(point1, point2) {
2   // dos dimensiones k esta definido al inicio
3
4   var distancia = 0;
5   for (var i = 0; i < k; i++) {
6     distancia += Math.pow((point1[i] - point2[i]), 2);
7   }
8   //devolver raiz de la sumatoria de las distancias
9   return Math.sqrt(distancia);
10 }

```

Listing 7: distanceSquared

2.5.1. Closest point brute force

Respecto para realizar la búsqueda por fuerza bruta es directamente comparar el punto a consultar con todos los puntos existentes en la lista completa generada, no se requiere de la iniciación de la estructura *kd-tree* sino que realiza el cálculo de las distancias clasificando la distancia menor cada vez que se realiza una comparación la distancia actual se toma como la mejor hasta que vaya recorriendo en todos los puntos hasta el final clasificando solamente el punto más próximo. El rendimiento del algoritmo es el mas ineficiente ya que requiere realizar comparaciones con todos los puntos.

```

1 function closest_point_brute_force(points, point) {
2   var distance = null;
3   var best_distance = null;
4   var best_point = null;
5   for (let i = 0; i < points.length; i++) {
6     distance = distanceSquared(points[i], point);
7     if (best_distance === null || distance < best_distance) {

```

```

8         best_distance = distance;
9         best_point = points[i];
10    }
11  } return best_point;
12 }
```

Listing 8: closestPointBruteForce

2.5.2. Naive closest point

Para la definición de esta función se requiere del ingreso del *kd-tree*, el punto a consultar y directamente ya se va actualizando la profundidad del árbol mientras va realizando su recorrido tomando su mejor resultado. Para este proceso siempre tomando el nodo actual desde el padre hacia los hijos, claramente tiene el mismo comportamiento del que se espera del binary tree, en este caso inicia tomando su primera dimensión, ya que solamente tenemos 2-dimensiones esta sera la X, luego realiza la consulta respecto a la misma dimensión del punto hacia donde tiene que dirigirse la consulta, actualizando el mejor nodo siempre con respecto al eje al que se consulta en su profundidad (va permutando realizando la consulta mediante la operación módulo %), finalmente si se eligió el camino correcto a seguir nuevamente se llama a la función principal recursivamente hasta encontrar el ultimo hijo respecto a la consulta. La contra de este algoritmo es que puede ser que el punto en realidad esta mas próximo en alguno de los ejes padre aun siendo este algoritmo mas eficiente con respecto al anterior

```

1 function naive_closest_point(node, point, depth = 0, best = null) {
2   if(node == null)
3     return best;
4
5   var axis = depth% k;
6
7   var best1 = null;
8   var camino = null;
9
10  if(best==null)
11    best1=node.point;
12
13  if(point[axis]>node.point[axis])
14    camino=node.right
15  else
16    camino=node.left
17  return naive_closest_point(camino,point,depth+1,best1)
18 }
```

Listing 9: naiveClosestPoint

2.6. Evalúe el resultado de las dos funciones implementadas anteriormente con este conjunto de datos

```

var data = [
  [40 ,70] ,
  [70 ,130] ,
  [90 ,40] ,
  [110 , 100] ,
  [140 ,110] ,
  [160 , 100]
];
var point = [140 ,90]; // query
```

Tomando ventaja de la función que grafica los puntos y la herramienta GraphViz se muestra en la figura 5

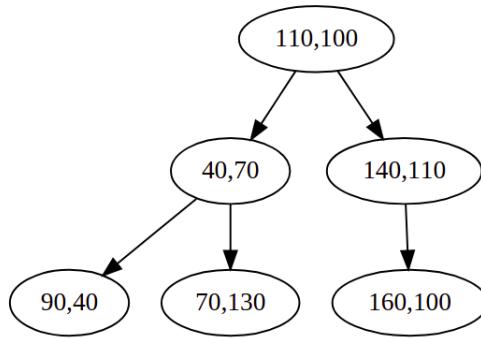


Figura 5: Vista GraphViz

Como se muestra en figura los resultados directos desde realizar la búsqueda del punto mas cercano utilizando la fuerza bruta da como resultado el punto verdaderamente más cercano pero con un rendimiento mayor, *closest_point_brute_force* directamente devuelve el resultado con respecto al acumulado en cada punto consultado. En cambio *naive_closest_point* va realizando las consultas de que lado del hijo se irá, en este caso tomara el lado derecho y seguirá recursivamente hasta hallar el hijo del total de la altura del árbol, tomando a este como el resultado final, siendo erróneo ya que el mínimo es su parent anterior a el por su cercanía al punto tal como se muestra en la figura 6.

```

▼ (2) [140, 110] □ "brute" ▼ (2) [160, 100] □ "naive"
  0: 140          0: 160
  1: 110          1: 100
  length: 2       2: 22.36
  ► __proto__: Array(0)  length: 3
                        ► __proto__: Array(0)
  
```

Figura 6: Resultado de ambos algoritmos de búsqueda del punto con menor distancia

2.7. Evalúe el resultado de las dos funciones implementadas anteriormente con este conjunto de datos

```

var data = [
  [40 ,70] ,
  [70 ,130] ,
  [90 ,40] ,
  [110 , 100] ,
  [140 ,110] ,
  [160 , 100] ,
  [150 , 30]
];
var point = [140 ,90]; // query
  
```

Tomando ventaja de la función que grafica los puntos y la herramienta GraphViz se muestra en la figura 7

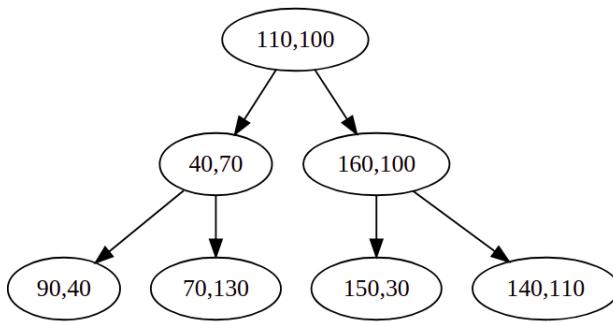


Figura 7: Vista GraphViz 2

closest_point_brute_force al igual que en el anterior caso, directamente devuelve el resultado con respecto al acumulado en cada punto consultado encontrando al punto con la distancia mas cercana al anterior. En cambio *naive_closest_point* va realizando las consultas de que lado del hijo se ira, en este caso tomara el lado derecho y seguirá recursivamente hasta hallar el hijo del total de la altura del árbol aun habiéndole aumentado un punto mas el cual acercaría mas su resultado pero el punto padre es el mas cercano dejando a este punto fuera del rango de respuesta próximo tal como se muestra en la figura 8.

```

▼ (2) [140, 110] ━━ "brute"
  0: 140
  1: 110
  2: 20
  length: 3
  ▶ proto : Array(0)

▼ (2) [150, 30] ━━ "naive"
  0: 150
  1: 30
  2: 60.83
  length: 3
  ▶ proto : Array(0)
  
```

Figura 8: Resultado del segundo conjunto de puntos con los algoritmos presentados

2.8. Ahora implemente la función closest point, siguiendo las recomendaciones dadas por el docente

Como mejora a la función *naive_closest_point* se realiza el cálculo de la distancia mas próxima durante la elección del punto mejor posicionado en el hijo donde se realizará la consulta, en este caso como ya se tiene previamente el mejor resultado el camino seguirá recorriendo tal cual como hacia pero el resultado se tomará respecto al punto que presentó la distancia mas cercana al punto a consultar.

```

1 function closest_point(node , point , depth = 0, best=null)
2 {
3   if(node==null) return best;
4
5   var axis=depth % k;
6   var camino = null;
7
8   if(best==null)
9   {
10     best1=node.point;
11   }
12   else if((distanceSquared(best, point)> distanceSquared(node.point,point)))
13     best1=node.point;
14   else
15     best1=best;
16
17   if(point[axis]>node.point[axis])
18     camino=node.right;
19
20   else
21     camino=node.left;
  
```

```

22     return closest_point(camino, point, depth+1, best1)
23 }

```

Listing 10: closestPoint

Utilizando como ejemplo el mismo resultado del ultimo conjunto de datos presentado se tiene el verdadero resultado con la distancia más próxima al punto a consultar como se muestra en la figura 9.

```

▼ (2) [140, 110] □      "brute" ▼ (2) [150, 30] □      "naive" ▼Node {point: Array(2), left: null, right: null, axis: 0} □ "closest point"
  0: 140                      0: 150                         axis: 0
  1: 110                      1: 30                          left: null
  2: 20                       2: 60.83                        ▼point: Array(3)
length: 3                     length: 3                         0: 140
                                ► __proto__: Array(0)           1: 110
                                ► __proto__: Array(0)           2: 20
                                length: 3                         ► __proto__: Array(0)
                                ► __proto__: Object            right: null
                                ► __proto__: Object

```

Figura 9: Mejora al algoritmo de búsqueda del punto más próximo a la consulta

2.9. Averigue e implemente una función KNN, que retorna los k puntos mas cercanos a un punto

Antes de implementar el algoritmo k-nearest-neighbour se precisa obtener una función que calcule el resultado más cercano, el cual recibe como parámetro el punto de consulta, el punto al que se esta aproximando y el punto del padre.

```

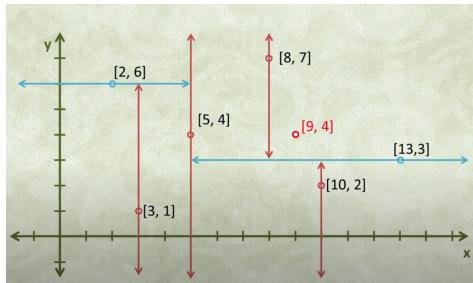
1 function masCercano(puntoConsulta, p1, p2) {
2   if (!p1) {
3     return p2;
4   }
5   if (!p2) {
6     return p1;
7   }
8   return (distanceSquared(puntoConsulta, p1) < distanceSquared(puntoConsulta, p2))?
9     p1 : p2;
}

```

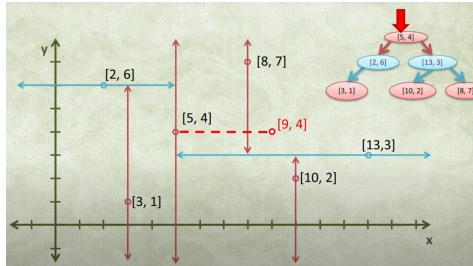
Listing 11: masCercano

Por consiguiente se define al algoritmo [1]:

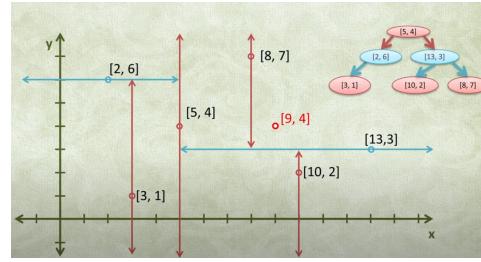
1. Comenzando con el nodo raíz, el algoritmo se mueve hacia abajo en el árbol de forma recursiva, de la misma manera que lo haría si se estuviera insertando el punto de búsqueda (es decir, va hacia la izquierda o hacia la derecha dependiendo de si el punto es menor o mayor que el nodo actual en la dimensión).
2. Una vez que el algoritmo llega a un nodo hoja, comprueba en ese punto de nodo si la distancia es mejor, ese punto de nodo se guarda como el mejor actual. (ver figura 10)
3. El algoritmo desenrolla la recursividad del árbol, realizando los siguientes pasos en cada nodo:
 - a) Si el nodo actual está más cerca que el mejor actual, entonces se convierte en el mejor actual.
 - b) El algoritmo verifica si podría haber puntos en el otro lado del plano de división que estén más cerca del punto de búsqueda que el mejor actual, se implementa una comparación simple para ver si la distancia entre la coordenada de división del punto de búsqueda y el nodo actual es menor que la distancia (coordenadas generales) desde el punto de búsqueda al mejor actual, en principio se definió la función de masCercano para realizar este proceso. Mientras el punto mas cercano sea menor al radio marcado entre la distancia del nodo



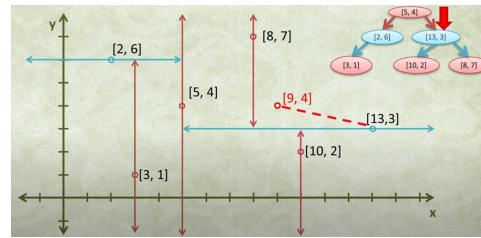
(a) El punto de consulta entra en el *kd-tree*



(c) Realiza la comparación de la distancia euclídea sobre el nodo raíz



(b) Realizar la búsqueda desde la raíz hacia las hojas



(d) Recorre el siguiente punto

Figura 10: Ingreso de un nuevo punto a consultar

anterior se tomara el siguiente hijo caso contrario se seguirá subiendo eliminando la rama que no cumple con el requisito (ver figura 11).

4. Cuando el algoritmo finaliza este proceso en el nodo raíz, la búsqueda está completa.

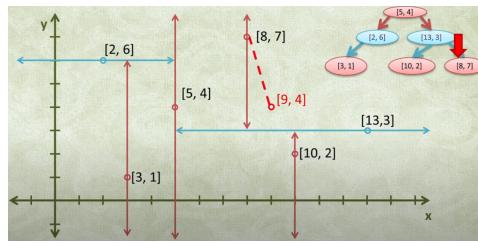
Al momento de repetir esta comparación tomará como máximo un tiempo de $O(2H)$ tomando H como la altura del árbol a recorrer, con esta condicional, la búsqueda en el árbol no realizará comparaciones con todos los puntos del árbol, puede solamente tomar $O(2H)$ como máximo de valores encontrados ordenados según su distancia.

En el siguiente código, se mantuvo la dimensión $k = 2$ solo con fines de utilizar solamente los puntos de dicha dimensión, para utilizarlo en otras implementaciones solamente debería cambiarse dicho valor por k . Igualmente como observación, por cada vez que se realiza el cálculo de la distancia con el punto a revisar, se ingresa a los vectores visitados el atributo distancia que no afecta a la revisión total de la k dimensión.

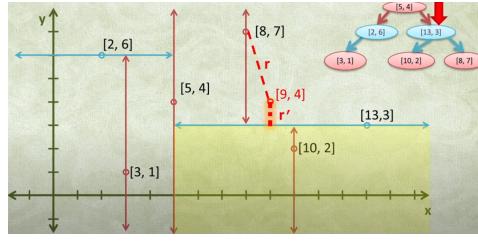
```

1 function knn(node, puntoConsulta, kpoints, depth = 0) {
2   if (!node) {
3     return null;
4   }
5
6   var temp;
7   var subTree1 = node.left;
8   var subTree2 = node.right;
9
10  if (puntoConsulta[depth % k] >= node.point[depth % k]) {
11    subTree1 = node.right;
12    subTree2 = node.left;
13  }
14
15  // Mejor distancia entre el padre y el subTree1.
16  masCercano(puntoConsulta, knn(subTree1, puntoConsulta, kpoints, depth + 1), node.
  point);

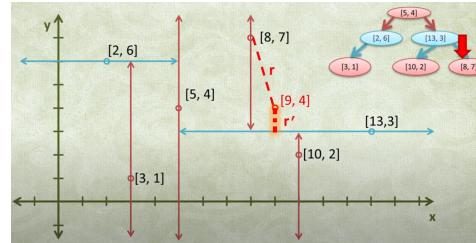
```



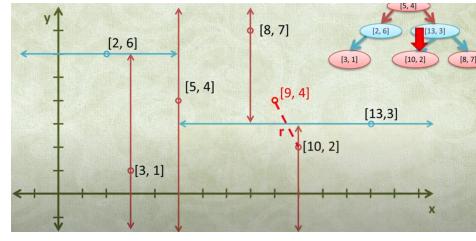
(a) Encuentra el ultimo punto de la rama del hijo mas próximo de la búsqueda



(c) Comparar la distancia del eje en dos valores tomando el menor para ir por la rama hija a consultar (r, r')



(b) La mejora del algoritmo, lo realiza desde retroceder recursivamente



(d) Si el radio marcado es menor al punto elegido se actualiza el menor punto cercano

Figura 11: Mediante el punto más cercano se ubica el punto a consultar

```

17
18 // Variable para ordenar por distancias
19 const sortByDistance = (a, b) => a[2] - b[2];
20
21 if (kpoints.length < vecinos) {
22   temp = node.point;
23   temp.push(Math.round(distanceSquared(puntoConsulta, temp)*100)/100);
24   kpoints.push(temp);
25   kpoints.sort(sortByDistance);
26 } else {
27   temp = node.point;
28   temp.push(Math.round(distanceSquared(puntoConsulta, temp)*100)/100);
29   if (temp[2] < kpoints[kpoints.length - 1][2]) {
30     kpoints.pop();
31     kpoints.push(temp);
32     kpoints.sort(sortByDistance);
33   }
34 }
35
36 if(kpoints.length < vecinos || kpoints[0][2] >= Math.abs(puntoConsulta[depth % k]
37 - node.point[depth%k])) {
38   masCercano(puntoConsulta, knn(subTree2, puntoConsulta, kpoints, depth +1), node.
39   point);
}
  
```

Listing 12: KNN

2.10. Implemente la función range query circle del KD-Tree

En primer lugar, se declara la función *closer_point* que recibe los parámetros del punto que chocará en los bordes de la consulta circular y también será reutilizada para la consulta rectangular. El resultado de realizar la comparación del punto que ingresa al rango de la consulta circular o rectangular

devolviendo inversamente el resultado ya que se quiere los puntos internos respecto a la distancia de los puntos que se estén comparando.

```

1  function closest_point(node,point_2,depth = 0)
2  {
3
4    if (node == null) return ;
5    //best = min(distanceSquared(node.point,point));
6    var nb = null;
7    var ob = null;
8    if (node.point[node.axis] > point_2[node.axis])
9    {
10      //if(node.left) best = min(best, naive_closest_point(node.left,point,depth++,best))
11      nb = node.left;
12      ob = node.right;
13    }
14    else
15    {
16      //if(node.right) best = min(best,naive_closest_point(node.right,point,depth++,best))
17      nb = node.right;
18      ob = node.left;
19    }
20
21    var best = closer_point(point_2,closer_point(point_2,closest_point(nb,point_2,depth+1),node),best);
22
23    if (distanceSquared(best.point,point_2) > Math.abs(point_2[node.axis] - node.point[node.axis]))
24    {
25      best2 = closer_point(point_2,closest_point(ob,point_2,depth+1),node);
26    }
27    best = closer_point(point_2,best2,best);
28
29
30    //if(best.estado == false) return;
31    return best;
32  }

```

Listing 13: closerPoint

La función *range_query_circle* recibe los parámetros de árbol *kd-tree*, el punto céntrico del círculo consulta, el tamaño del radio, el acumulador de los puntos visitados y finalmente la profundidad del árbol que se irá acumulando. En el recorrido del cada vez que se dirige el punto céntrico respecto al radio, siempre inicia con la revisión desde el nodo raíz hacia las ramas, comparando el eje en el que se realiza la primera consulta hacia los hijos derecho o izquierdo. al tener la dirección se realiza nuevamente la consulta recursiva respecto al lado del hijo hallado, como en el caso del algoritmo KNN, si el punto respecto al radio principal es mas corto al punto del hijo donde esta apuntado la distancia, se balancea y se toma la consulta del punto que esta en otra rama y directamente se hace la comparación si la distancia calculada al punto principal es menor al radio se inserta en el arreglo consulta. finalmente actualizamos el mejor punto recursivamente. Cada consulta de hallar el punto más cercano, utilizamos la función *closer_point* explicada al principio.

```

1  function range_query_circle ( node , center , radio , queue , depth = 0 ){
2    if (node==null) return null;
3
4    var axis = node.axis ;
5    var nb = null;
6    var ob = null;
7
8    if (center[axis] < node.point[axis]){
9      nb=node.left;

```

```

10     ob=node.right;
11   } else {
12     nb=node.right;
13     ob=node.left;
14   }
15
16   var best=closer_point(center,node,range_query_circle(nb,center,radio,queue,depth+1))
17   ;
18
19   if(Math.abs(center[axis]-node.point[axis]) <= radio || distanceSquared(center,best.
20     point) > Math.abs(center[axis]-node.point[axis])){
21
22     if(distanceSquared(center,node.point) <= radio){
23
24       queue.push(node.point);
25     }
26
27     best=closer_point(center,best,range_query_circle(ob,center,radio,queue,depth+1));
28   }
29
30   return best ;
31 }
```

Listing 14: rangeQueryCircle

Complementando en la consulta gráfica, para distinguir la consulta circular, en el inicio de la grafica se toma la distancia invertida del eje y para mejor visualización y distinción del centro de la figura respecto al eje. Así mismo se muestra la consulta esta englobando a los puntos que están dentro del círculo como en la figura 12

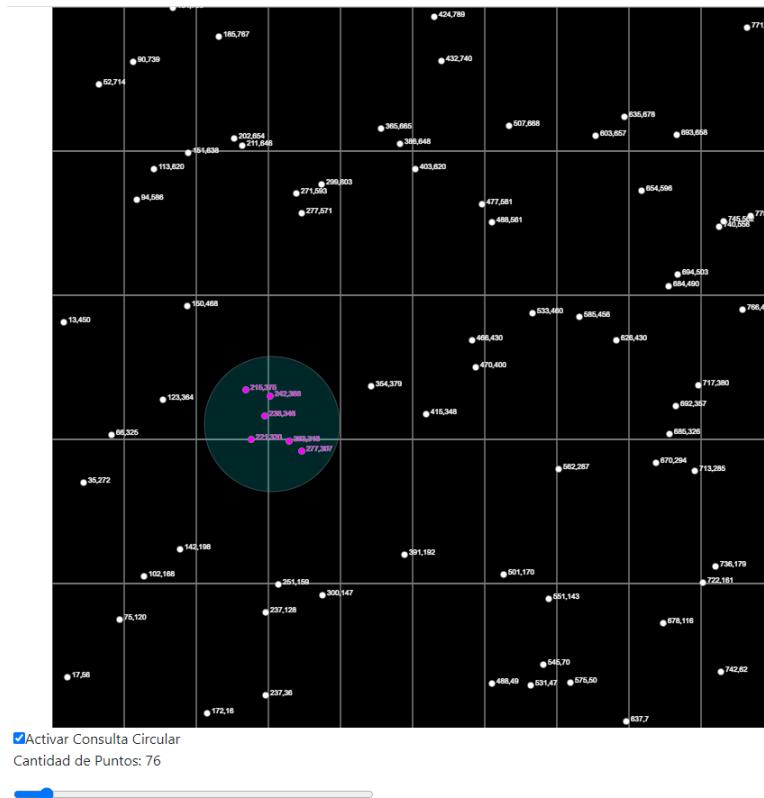


Figura 12: rangeQueryCircle

2.11. Implemente la función range query rec del KD-Tree, esta vez el range representa un rectángulo

La función *range_query_rect* recibe los parámetros de árbol *kd-tree*, el punto céntrico del círculo consulta, el tamaño del ancho y alto en un par ordenado, el acumulador de los puntos visitados y finalmente la profundidad del árbol que se irá acumulando. Para este caso la revisión ya no se hace mediante la consulta de un radio sino, a partir de dos parámetros, ancho*2 y alto*2, igualmente la primera parte inicia la consulta desde la raíz para ir descendiendo hacia los hijos, en este caso para comparar el ingreso del punto al rectángulo se realiza una primera comparación sobre el eje en el cual se está evaluando el punto en el nodo actual respecto al eje del rectángulo, si es menor o igual para estar incluido en la consulta inmersa del rectángulo, igualmente verificar si el punto está dentro del rango de la distancia respecto al padre en el mismo eje para que así se pueda tomar en cuenta los puntos que no pertenecen directamente al resultado del hijo inmediato. Nuevamente se realiza la búsqueda del punto más cercano recursivamente al punto que va descendiendo en el árbol.

```

1  function range_query_rect ( node , center , hug , queue , depth = 0 ){
2    if (node==null) return null;
3
4    var axis = node.axis ;
5    var nb = null;
6    var ob = null;
7
8    if (center[axis]<node.point[axis]){
9      nb=node.left;
10     ob=node.right;
11   } else {
12     nb=node.right;
13     ob=node.left;
14   }
15   var best=closer_point(center,node,range_query_rect(nb,center,hug,queue,depth+1));
16
17   if(Math.abs(center[axis]-node.point[axis])<=hug[axis]*2 || distanceSquared(center,
18     best.point)>Math.abs(center[axis]-node.point[axis])){
19     if(Math.abs(center[0]-node.point[0])<=hug[0] && Math.abs(center[1]-node.point[1])
20       <=hug[1]){
21
22       queue.push(node.point);
23     }
24     best=closer_point(center,best,range_query_rect(ob,center,hug,queue,depth+1));
25   }
26
27   return best ;
}

```

Listing 15: rangeQueryRect

Complementando en la consulta gráfica, igualmente con la consulta circular se realizó sobre la misma cantidad de puntos en este caso se debe comparar la distancia de los ejes respecto al tamaño del rectángulo en el eje que se compara, dando como resultado los puntos que estarán inmersos dentro del rectángulo consulta 13

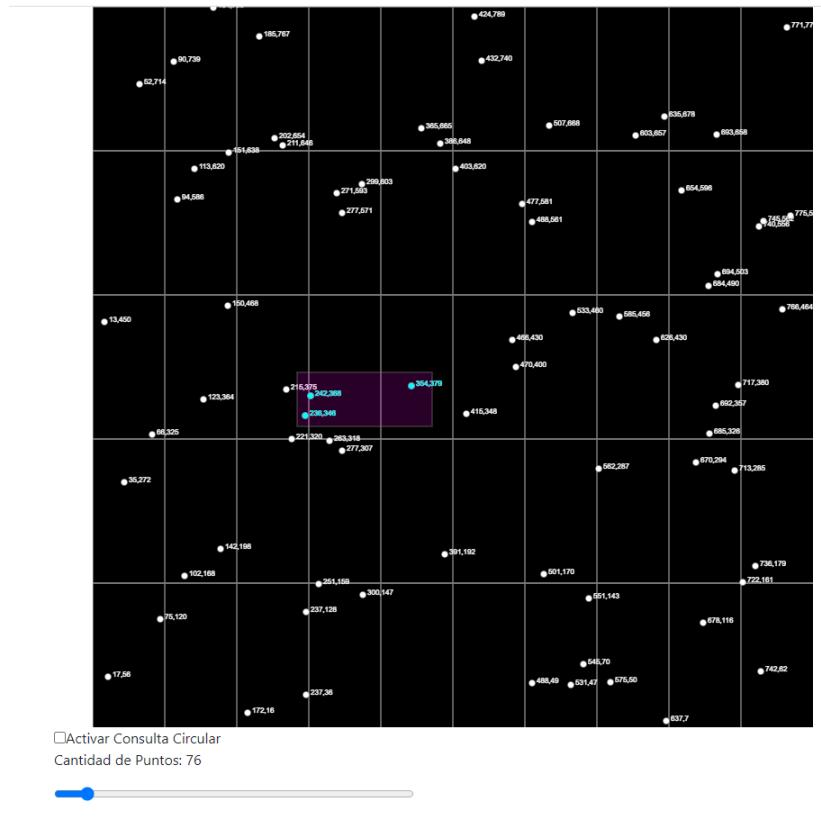


Figura 13: rangeQueryRect

3. Trabajo de investigación

La estructura KD-Tree es una estructura multidimensional de k dimensiones. Esta permite implementar búsquedas por similitud como *KNearestNeighbor* o *Closestpoint*. Adicionalmente, se puede usar esta estructura como un clasificador. Usted debe implementar este clasificador en el tema de su preferencia. A continuación detallamos el algoritmo:

Algorithm 1 KNN Clasifier

Input: X : training data; y : objecto to be classified

Output: Classification for y

Extract features of each sample;

Build KD-Tree;

 Select KNN of y in X ;

 $\text{Class}(y) \leftarrow \max$ of classes (k closest objects)

3.1. Dataset Elegido

Para el dataset, usamos Kaggle, que es una plataforma en la nube de Data Science más grande del mundo. Donde nos permite encontrar y publicar una amplia variedad de dataset. Con respecto a la elección del dataset, directamente se revisaron muchas otras opciones como el generador del buscador de películas en general, en este caso solamente se cuenta con la información de 3323 Películas en el catálogo de Netflix respecto a la cantidad presentado en el dataset *Netflix_Movies* [3] (ver figura

14). El dataset esta compuesto por 10 columnas(índice, nombre de la película, duración, año, género, director, actores, país, rating, ingreso en netflix), las cuales describen a las películas mencionadas anteriormente, el dataset esta en formato CSV cuyo plan inicial es realizar su lectura con ayuda de la librería Pandas de Python para realizar el análisis de preprocesamiento de la información respecto a las columnas que se tomarán como característica principal, pero para la implementación directa del vector característico se paso a un formato JSON.

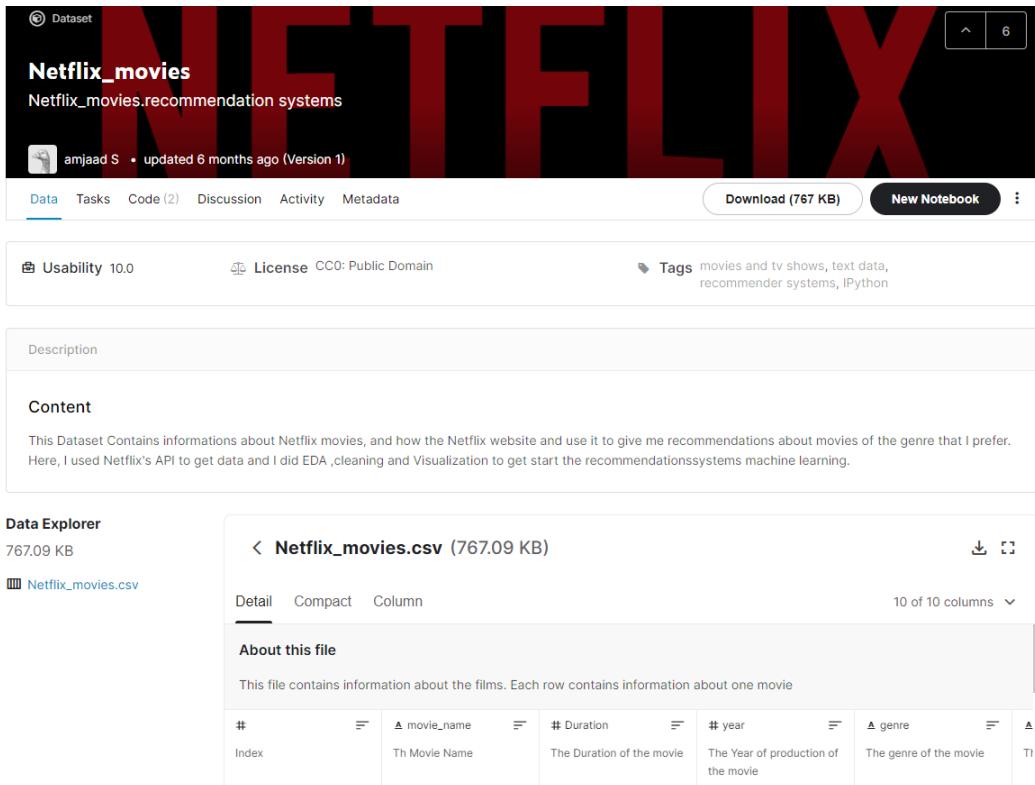


Figura 14: Kaggle *Netflix_Movies* Dataset

3.2. Extracción de Características

Para la extracción de características, usamos Colab, que es un servicio de Cloud que nos permite programar y ejecutar código de Python, ideal para estudiantes, científico de datos o investigador en Inteligencia Artificial. Empleamos el modelo Bag of words(bolsa de palabras) es un método que se utiliza en el procesado del lenguaje para representar documentos ignorando el orden de las palabras.

1. Conectamos desde Colab con kaggle para importar el dataset almacenado en kaggle.
2. Generamos un vector característico. Para este proceso elegimos el campo género, ya que obtuvo una dimensión de 29 en comparación con el resto de campos. Adicional a ello el vector característico final se obtuvo adicionando los 29 valores del bag of words, conjuntamente con los campos duración, año y rating, dando una dimensión de 32 valores del vector característico.

Columna	Bag of Words
Género	29
País	101
Director	3989
Nombre de la película	4569
Actor	17865

Figura 15: Bag of words - comparación de dimensiones

3. Exportamos el objeto generado en JSON, ya que nuestras implementaciones del kd-tree y knn están implementadas en Javascript.

Listing 16: Objeto JSON

3.3. Aplicación Web para las Películas

Con el objetivo de ver el funcionamiento de las operaciones del KDtree y el proceso de busqueda usando el algoritmo de Knn, se determino usar la base de caracteristicas del dataset de Netflix dado en Kaggle <https://www.kaggle.com/amjaads/netflix-movies>, para ello se determino usar la base de pandas <https://pandas.pydata.org/> para realizar las consultas y filtros sobre los datos, y puedan estos se retornados a la interfaz.

En este caso la comunicación entre Python y Javascript fue dada gracias a el uso Flask <https://flask.palletsprojects.com/en/2.0.x/> el cual permite desarrollar una interfaz web usando Python.

La parte de preparación de características se observa en la figura 17, donde extrae las 3 características numéricas (duración, año, calificación) más 37 características pertenecientes al género de la película, el cual se obtuvo al aplicar el CountVectorizer sobre la columna Genero.

```
1 def buildJsonSelectedMovie(vectGenTotal, selectMovie):  
2  
3     #conocer el index del dataframe  
4     indexDfMovie = selectMovie.index[0]
```

```

5      #datos textuales
6      colGenero = selectMovie[ "genre" ]
7      featuresGenero = countvec.fit_transform( colGenero )
8
9
10     dataMovie = selectMovie.to_json(orient="records")
11     parsedObjMovie = json.loads(dataMovie)
12
13     #formando el json data
14     datajson = { "data": [] }
15
16     #recorremos cada registro de data y seleccionar especificas columnas
17     numericData = [selectMovie.loc[indexDfMovie][ "Duration" ], selectMovie.loc[ indexDfMovie ][ "year" ], selectMovie.loc[indexDfMovie][ "rating" ]]
18
19     datajson[ "data" ].append({
20         "vector": numericData + vectGenTotal[indexDfMovie].tolist(),
21         "obj": parsedObjMovie[0]
22     })

```

Listing 17: getFeatureVectors.py

La parte de consulta de los datos usando Pandas dado en la figura 18. Por ejemplo, para un id particular de película seleccionada por el usuario y con la debida preparación de características dada en la función "buildJsonSelectedMovie", permitiendo así devolver un dato como objeto Json para la interfaz.

```

1      #mandamos el json del total de pelicula para el kdtree
2      vectGenTotal ,totalMovies_json = buildJsonEntireData(movieData)
3      idMovie = request.form.get("col4");
4      #consultamos mediante el id la pelicula seleccionada
5      selMovie = movieData[movieData[ "Unnamed: 0" ] == int(idMovie)]
6      selMovie_json = buildJsonSelectedMovie(vectGenTotal,selMovie)
7      return jsonify(result=[selMovie_json, totalMovies_json]);

```

Listing 18: consultaMovies.py

La parte del uso del KDTree y la consulta con KNN es dada en la figura 19 donde se consigue lo enviado por la consulta en Python, esto permite construir el árbol KDTree con el set completo de películas y realizar la consulta con la película seleccionada por el usuario, ambos ya fueron vectorizados y adicionando su objeto al interior del nodo. Finalmente servir a la interfaz almacenando en el localStorage del navegador.

```

1      function mostrarResultado(data){
2          $("#result").text(data.result[0]);
3
4          //las 2 fuentes de datos en formato json
5          var dataSelMovie = JSON.parse(data.result[0]);
6          var dataTotalMovies = JSON.parse(data.result[1]);
7
8          console.log("Data el primero",dataTotalMovies[ "data" ][0]);
9          root = build_kdtree(dataTotalMovies[ "data" ]);
10         console.log("Root",root);
11         kvecinos = [];
12         resultNodes = [];
13         knarestpoints(root,dataSelMovie[ "data" ][0][ "vector" ],kvecinos,resultNodes,depth
14         =0);
15         console.log("KVECINOS:",kvecinos);
16         $("#result_vecinos").text(kvecinos);
17
18         //guardar en el local storage inicio de búsqueda
19         window.localStorage.setItem("resultNodes",JSON.stringify(resultNodes));

```

Listing 19: viewmovie.html

3.4. Vista Detalle de Películas

Luego de realizar la búsqueda y darnos como resultado un archivo JSON necesitamos interpretar este archivo y generar un escenario de visualización de los resultados, para esto desde el backend se ha guardado el json en el almacenamiento 'local storage' el cual puede ser abierto desde el frontend, específicamente desde javascript, la sentencia que realiza esta acción es la siguiente.

```

1   var dataLocalStorage = JSON.parse(window.localStorage.getItem("resultNodes"));
2   var kvecinosVector = JSON.parse(window.localStorage.getItem("kvecinosVector"));
3
4
5   console.log("LocalStorage - Nodos Encontrados:", dataLocalStorage[0]);
6   console.log("LocalStorage - Nodos Encontrados:", kvecinosVector[0][33]);
7
8   for(var i=0; i<dataLocalStorage.length; i++){
9     data_nom.push(dataLocalStorage[i].obj.movie_name);
10    data_dur.push(dataLocalStorage[i].obj.Duration);
11    data_yea.push(dataLocalStorage[i].obj.year);
12    data_gen.push(dataLocalStorage[i].obj.genre);
13    data_dir.push(dataLocalStorage[i].obj.director);
14    data_act.push(dataLocalStorage[i].obj.actors);
15    data_cou.push(dataLocalStorage[i].obj.country);
16    data_rat.push(dataLocalStorage[i].obj.rating);
17    data_ent.push(dataLocalStorage[i].obj.enter_in_netflix);
18  }
  
```

Listing 20: vistaLocalStorage.js

Ya teniendo todos los elementos de la búsqueda, calculamos los parámetros de valoración que mostraremos de forma gráfica en barras y datos numéricos.

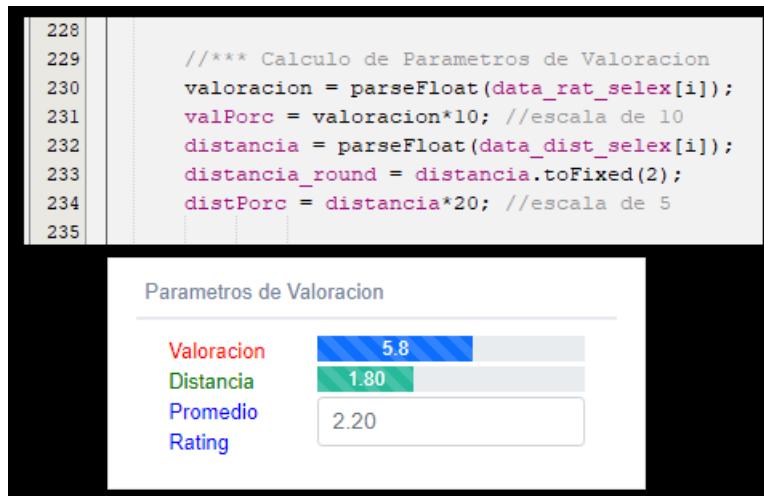


Figura 16: Parámetros de Valoración

Finalmente las características de cada película son mostradas en paneles de visualización descendentes, cada uno contiene un video embebido de youtube que por defecto es la intro de netflix, a cada panel mostrado se le agregó un botón 'Buscar' que permite ir directamente a youtube con la búsqueda del trailer correspondiente, también se tiene un botón 'Agregar' que permite insertar la URL del tráiler en el sistema, el resultado final sería el siguiente.

Algoritmo KD-Tree Knn Netflix (grupo 9)

Detalles de Películas

Parametros de Valoracion	Valoracion	Distancia	Promedio	Rating
	8.1		3.24	

#Selfie
Valoracion: 6.1

Duracion: 125
Year: 2014
Genero: Comedies, Dramas, International Movies
Director: Cristina Jacob
Pais: Romania
Rating: 6.1
Enter: June 1, 2019
Trailer: [Buscar](#) [Agregar](#)



Inkaar
Valoracion: 6

Duracion: 125
Year: 2013
Genero: Dramas, International Movies, Romantic Movies
Director: Sudhir Mishra
Pais: India
Rating: 6
Enter: July 5, 2020
Trailer: [Buscar](#) [Agregar](#)



Gaddar: the Traitor
Valoracion: 6.3

Duracion: 126
Year: 2015
Genero: Dramas, International Movies, Thrillers
Director: Amitoy Martin
Pais: India
Rating: 6.3
Enter: November 1, 2018
Trailer: [Buscar](#) [Agregar](#)



John Day
Valoracion: 5.7

Duracion: 125
Year: 2013
Genero: International Movies, Thrillers
Director: Abhishek Solomon
Pais: India
Rating: 5.7
Enter: December 31, 2019
Trailer: [Buscar](#) [Agregar](#)



Dharam Sankat Mein
Valoracion: 6.3

Duracion: 127
Year: 2015
Genero: Comedies, International Movies
Director: Fuwad Khan
Pais: India
Rating: 6.3
Enter: July 5, 2020
Trailer: [Buscar](#) [Agregar](#)



[Ver 5 mas ...](#)

Figura 17: Vista Gráfica KD Tree KNN Netflix

Referencias

- [1] H. Samet, *Foundations of multidimensional and metric data structures*. Morgan Kaufmann series in data management systems, Academic Press, 2006.
- [2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second ed., 2000.
- [3] amjaad S, “Netflix movies.” <https://www.kaggle.com/amjaads/netflix-movies>, 2021. [Netflix movies.recommendation systems].

Listings

1.	index.html	1
2.	kdtree.js	2
3.	buildKdTree	2
4.	getHeight	3
5.	generateDot	4
6.	sketch.js	5
7.	distanceSquared	6
8.	closestPointBruteForce	6
9.	naiveClosestPoint	7
10.	closestPoint	9
11.	masCercano	10
12.	KNN	11
13.	closerPoint	13
14.	rangeQueryCircle	13
15.	rangeQueryRect	15
16.	Objeto JSON	18
17.	getFeatureVectors.py	18
18.	consultaMovies.py	19
19.	viewmovie.html	19
20.	vistaLocalStorage.js	20