

BALL TREE y KD-TREE CASO BÚSQUEDA DE IMÁGENES

Patrick Lazo Ruben Torres Lima

January 2018

1. Introducción

La búsqueda de imágenes inversa le permite utilizar como entrada una imagen y encontrar imágenes visualmente similares.

Para poder lograr esto se tiene que seguir los siguientes pasos.

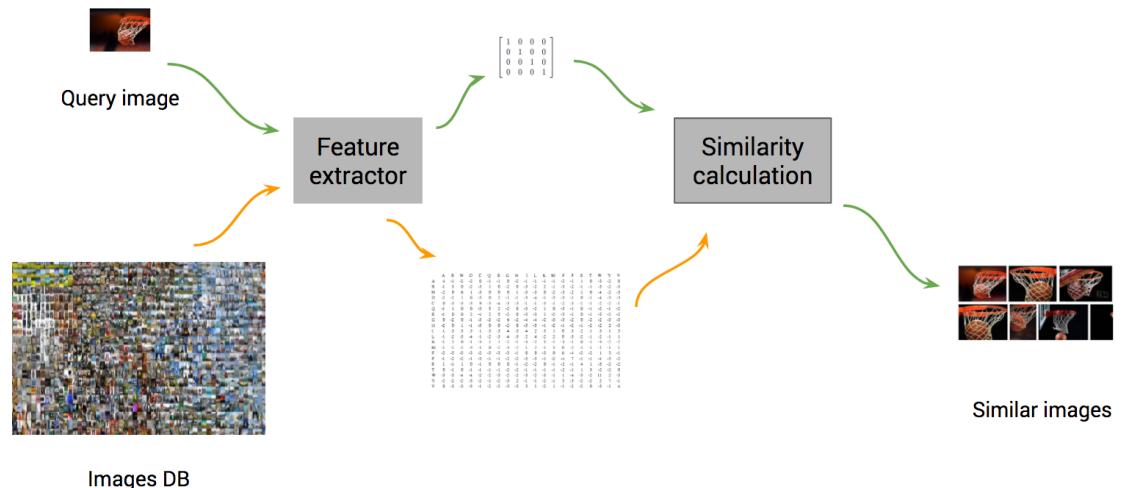


Figura 1: Metodología para la recuperación de imágenes

1.1. Extracción de Características

En este paso se busca convertir la imagen en un vector característico habiendo varios métodos como:

- Histogramas

- Convolución

Lo que se busca en este paso es poder representar en números de la mejor manera posible la imagen.

1.2. Calculo de Similitud

Una vez obtenido nuestro vector característico tenemos que hacer la búsqueda de similitud de nuestro imagen consultada con toda nuestra base de datos.

En esta parte se usan estructuras espaciales para poder conseguir un costo de búsqueda mas eficiente como bktree,balltree.

2. Ball Tree

2.1. Implementación

Ball Tree es un árbol métrico. En su forma original, para cada uno de los puntos del nodo son asignandos al centro más cercano de los dos hijos del nodo. Los hijo se eligen para tener la distancia máxima entre ellos. Para la construcción del árbol primero, se localiza el centroide de los puntos, y el punto con la mayor distancia de este centroide se elige como el centro del primer hijo. El centro del segundo hijo se elige como el punto más alejado del primero.

```
function construct_balltree is
    input:
        D, an array of data points
    output:
        B, the root of a constructed ball tree
    if a single point remains then
        create a leaf B containing the single point in D
        return B
    else
        let c be the dimension of greatest spread
        let p be the central point selected considering c
        let L,R be the sets of points lying to the left and right of the median along dimension c
        create B with two children:
            B.pivot = p
            B.child1 = construct_balltree(L),
            B.child2 = construct_balltree(R),
            let B.radius be maximum distance from p among children
        return B
    end if
end function
```

Figura 2: Construcción de Ball Tree [1]

```

Procedure BallKNN ( $PS^{in}$ ,  $Node$ )
begin
  if ( $D_{min}^{Node} \geq D_{sofar}$ ) then /* If this condition is satisfied, then impossible
    for a point in Node to be closer than the
    previously discovered  $k^{th}$  nearest neighbor.*/
    Return  $PS^{in}$  unchanged.
  else if (Node is a leaf)
     $PS^{out} = PS^{in}$ 
     $\forall x \in Points(Node)$ 
    if ( $|x - q| < D_{sofar}$ ) then /* If a leaf, do a naive linear scan */
      add  $x$  to  $PS^{out}$ 
      if ( $|PS^{out}| == k + 1$ ) then
        remove furthest neighbor from  $PS^{out}$ 
        update  $D_{sofar}$ 
    else /*If a non-leaf, explore the nearer of the two
       $node_1$  = child of Node closest to  $q$  child nodes, then the further. It is likely that
       $node_2$  = child of Node furthest from  $q$  further search will immediately prune itself.*/
       $PS^{temp} = BallKNN(PS^{in}, node_1)$ 
       $PS^{out} = BallKNN(PS^{temp}, node_2)$ 
  end

```

Figura 3: Busqueda dentro de Ball Tree [2]

3. KD-TREE

3.1. Implementación

Si quisieramos calcular la distancia del vector característico de la imagen consultada con toda la base de datos lo primero que se nos vendría a la mente sería compararlo uno por uno y devolver los más cercanos (los más parecidos visualmente) lo que nos daría un costo $O[ND^2]$ para N consultas en D dimensiones.

Un mejor enfoque es usar KD-TREE lo que reduce número de cálculos de distancia mediante la agregación de manera eficiente de la información.

La idea básica es que si el punto A está muy distante del punto B , y el punto B es muy cerca del punto C , entonces sabemos que los puntos A y C son muy distantes, sin necesidad de calcular explícitamente su distancia.

Por consiguiente, el coste computacional se puede reducir a $O[DN\log(N)]$, con un coste de consulta de $O[\log(N)]$.

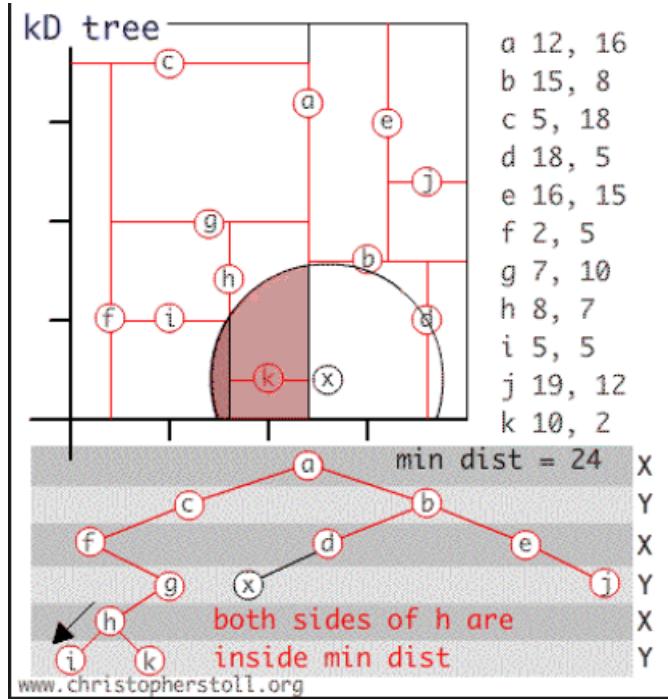


Figura 4: Búsqueda del vecino más cercano con kd-tree

El único problema es que bktree tiende ha ser mas eficiente cuando el número de dimensiones es menos por lo que si nuestro vector característico fue creado usando histogramas tendríamos 256 dimensiones.

4. Experimentos

Para los experimentos se uso

- INRIA Holidays Dataset con un total de 805 imágenes cada una con 1023 datos caracteristicos.
- Una computadora core i7 de cuarta generacion.
- El lenguaje de programacion es python.

Basados en el los resultados mostrados en la figura 5, podemos decir que *Ball Tree* tiene a demorarse en construir el arbol con 805 imágenes dentro de 0.10 seg y 0.125 seg. mostrandonos una gran velocidad para la construccion del mismo arbol. Segun [2] nos demuestra que *Ball Tree* es una estructura metrica con una velocidad media de construccion.

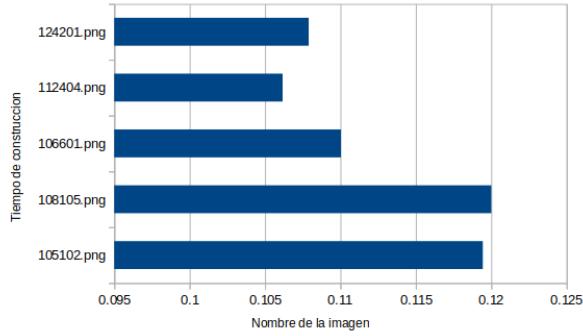


Figura 5: Diagrama de columna basado en las pruebas de construccion de Ball Tree, tiempo de construccion en segundos y construccion basado en la imagen que se buscara

Cuando hablamos de la funcion principal del árbol nos referimos a la velocidad con la que este puede realizar sus busqueda con el diagrama de dispersion mostrada en la figura 6 tomando en cuenta que el número de datos para compararse por imagen es de 1023 la busqueda para sus vecinos mas cercanos es muy rápida comparandose a otros arboles [3]

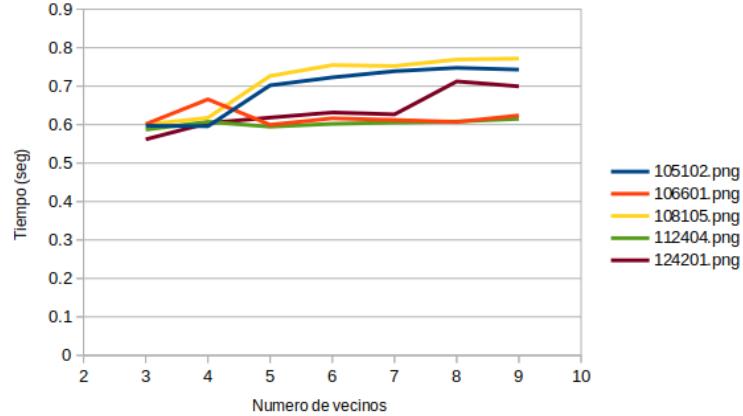


Figura 6: Diagrama de dispersion para busqueda de Ball Tree, tiempo de busqueda y numero de vecinos buscados

Cuando hablamos de *performance* *Ball Tree* nos da buenos resultados. Basandonos en la cantidad máxima de imagenes similares. Comparado a otras estructuras *Ball Tree* tiene mejores resultados [4]



Figura 7: Resultados de busquedas con 3 vecinos

5. Conclusiones

Ball Tree siendo una de las variacion de *KD tree* obtiene mejores resultados cuando es comparado en base a la velocidad de busqueda y su *performance*, Cuando hacemos referencia a la construccion de la estructura este tiende a ser un poco menos eficiente que otras estructuras, pero tiene buenos resultados.

El código fuente puede ser accedido en:

BALL TREE: <https://github.com/RubenJTL/Ball-TreeV1>

BK TREE: <https://github.com/patrick100/Bktree-Image-Retreival>

Referencias

- [1] Omohundro, S. M. (1989). Five balltree construction algorithms (pp. 1-22). Berkeley: International Computer Science Institute.
- [2] Liu, T., Moore, A. W., & Gray, A. (2006). New algorithms for efficient high-dimensional nonparametric classification. *Journal of Machine Learning Research*, 7(Jun), 1135-1158.
- [3] Kibriya, A. M., & Frank, E. (2007, September). An empirical comparison of exact nearest neighbour algorithms. In *PKDD* (Vol. 7, pp. 140-151).

- [4] Kumar, N., Zhang, L., & Nayar, S. (2008). What is a good nearest neighbors algorithm for finding similar patches in images?. Computer Vision–ECCV 2008, 364-378.
- [5] Reverse Image Search, available:<https://shuaiw.github.io/2017/10/05/reverse-image-search.html>
- [6] Implementing kd-tree python, available:<https://www.datasciencecentral.com/profiles/blogs/implementing-kd-tree-for-fast-range-search-nearest-neighbor>