IIC2440 – Procesamiento de Datos Masivos

Locally-Sensitive Hashing

1. Motivacion y definiciones

La técnica de Locally-sensitive Hashing (LSH) intenta resolver el siguiente problema.

> Dada una colección de elementos y una noción de distancia, retornar los pares de elementos más cercanos de acuerdo a esa distancia.

Naturalmente, podríamos calcular esta distancia para cada par de elementos, revisando los más cercanos. Pero esto implica una cantidad cuadrática de operaciones, que es algo inpensable para una base de datos.

La forma en que resolvemos esto con LSH es usando funciones de hash que cumplan con la siguiente propiedad: los hashes de elementos más cercanos tienen una probabilidad alta de ser iguales, y los hashes de elementos lejanos tienen una probabilidad baja de ser iguales.

Recordemos que una distancia es una función d sobre pares de elementos de un conjunto, que cumple con las siguientes propiedades:

- d es simétrica.
- $d(x, x) = 0$
- $d(x, y)$ es siempre positiva
- Satisface la desigualdad triangular: $d(x, z) \leq d(x, y) + d(y, z)$

Para efectos de esta definición suponemos un conjunto E de elementos, equipado con una distancia d. A esto se le conoce normalmente como un *espacio métrico*.

Definición. Una familia F de funciones $E \to \mathbb{R}$ es (d_1, d_2, p_1, p_2)-sensitiva si para cada par de elementos x e y, y una función $f \in F$ tomada aleatoriamente, tenemos que:

- Si $d(x, y) \leq d_1$, entonces la probabilidad de que $f(x) = f(y)$ es mayor o igual a p_1
- Si $d(x, y) \geq d_2$, entonces la probabilidad de que $f(x) = f(y)$ es menor o igual a p_2

¿Para que sirve esto? Podemos usar la siguiente receta para (intentar) tomar todos aquellos pares de elementos tales que $d(x, y) \leq k$:

- calculamos $f(e)$ para cada elemento $e \in E$.
- para cada par de elementos (x, y) tal que $f(x) = f(y)$, calculamos $d(x, y)$. Si es menor o igual a k, lo reportamos.

Ejercicio. Supongamos que queremos reportar pares de elementos tales que $d(x, y) \leq k$, y usamos la receta descrita arriba para una función (k, d_2, p_1, p_2)-sensitiva. Qué tipo de error podría aparecer? con que probabilidad aparecen?

Claramente no tenemos falsos positivos, pues nunca vamos a reportar nada que tenga distancia mayor a k. Pero podría ser que tengamos falsos negativos: pares con $d(x, y) \leq k$ pero tales que $f(x) \neq f(y)$. De acuerdo a la definición de arriba, esto solo puede pasar con probabilidad $1 - p_1$. En otras palabras, si queremos que la probabilidad de tener falsos negativos sea menor o igual a p_{fp}, necesitamos una familia de funciones que sea $(k, k', 1 - p_{fp}, p')$-sensitiva. En este ejemplo, el valor de k' y p' nos permite modular cuan costosa es nuestra búsqueda: mientras más pequeña es k' y más pequeña es p', vamos a tener menos pares donde $f(x) = f(y)$.

2. Distancia angular

Vamos a ver un esquema de LSH para la distancia angular, definida para espacios euclidianos y otros espacios vectoriales. Esta distancia se basa en la llamada *similaridad de coseno*, que mide el ángulo entre dos vectores.

2.1. Definiendo distancia angular

Formalmente, dados vectores $\bar{u}$ y $\bar{v}$, la similaridad de coseno nos entrega el coseno del ángulo entre $\bar{u}$ y $\bar{v}$. Acá para $\bar{x} = (x_1, \ldots, x_n)$ la norma $||\bar{x}|| = \sqrt{x_1^2 + \cdots + x_n^2}$ es la norma euclidiana usal, aunque podemos definir esta distancia usando también otras normas.

$$\text{sc}(\bar{u}, \bar{v}) = \frac{\bar{u} \cdot \bar{v}}{||\bar{u}||||\bar{v}||}.$$

La similaridad de coseno nos da un valor entre -1 y 1, por lo que no es una distancia. Para transformarla en una distancia, tomamos el arco coseno de ese valor (en grados):

$$d_a(\bar{u}, \bar{v}) = \arccos(\text{sc}(\bar{u}, \bar{v})).$$

La distancia angular es una buena distancia para vectores en los que nos importan más las direcciones que las magnitudes, y también es usada en casos donde la dimensión de los vectores es muy alta. Notar que una distancia de 0 indica que dos vectores tienen la misma dirección (osea uno es una ponderación del otro por un escalar, lo que para nuestros fines indica que es el mismo elemento). Por el contrario, una distancia de 180 nos indica que los vectores tienen direcciones opuestas.

2.2. Esquema de Locally Sensitive Hashing para distancia angular

Definimos entonces a F como la familia $f_{\bar{w}}$de funciones dada por vectores aleatorios $\bar{w}$, en donde $f_{\bar{w}}(\bar{u}) = f_{\bar{w}}(\bar{v})$ si y solo si $\bar{w} \cdot \bar{u}$ y $\bar{w} \cdot \bar{v}$ tienen el mismo signo.

Calculando las propiedades de este esquema Notar que los vectores $\bar{u}$ y $\bar{v}$ siempre van a definir un plano (el ángulo entre ellos de hecho se mide en ese plano).

Tomemos ahora un vector $\bar{w}$ cualquiera, y sea p el hiperplano definido por los puntos cuyo producto con $\bar{w}$ es 0 (es decir, w es un vector normal de ese plano). Ahora podrían pasar varias cosas.

- Los productos $\bar{u} \cdot \bar{w}$ y $\bar{v} \cdot \bar{w}$ tienen distinto signo. Esto significa que $\bar{w}$ queda entre medio de $\bar{u}$ y $\bar{v}$ (es decir, la recta que define la intersección entre p y el plano definido por $\bar{u}$ y $\bar{v}$ está dentro del ángulo que definen $\bar{u}$ y $\bar{v}$.

- Tanto $\bar{u} \cdot \bar{w}$ y $\bar{v} \cdot \bar{w}$ tienen el mismo signo. En este caso, el hiperplano p intersecta con el plano de $\bar{u}$ y $\bar{v}$ en algun lugar fuera del ángulo definido por $\bar{u}$ y $\bar{v}$.

¿Cual es la probabilidad del primer caso si elegimos $\bar{w}$ al azar? Notar que, en un caso general[1], p intersecta al plano formado por $\bar{u}$ y $\bar{v}$ como una recta. Como esa recta p podría ubicarse en cualquier lugar dentro o fuera del ángulo que definen $\bar{u}$ y $\bar{v}$ con igual probabilidad, si $d_a(\bar{u}, \bar{v}) = \theta$, entonces el primer caso se va a dar con probabilidad $\theta/180$. Claramente, mientras más alta es la distancia angular, mayor es la probabilidad de que el producto de $\bar{u}$ y $\bar{v}$ con $\bar{w}$ tengan distinto signo.

Entonces, para cualquier $0 \leq d_1 < d_2 \leq 180$), F es una familia $(d_1, d_2, (180-d_1)/180, (180-d_2)/180)$-sensitiva.

Veamos esto. Supongamos primero que $(\bar{u}, \bar{v}) \leq d_1$. Entonces para cualquier vector aleatorio $\bar{w}$, vimos que la probabilidad de que tengan distinto signo es $d_a(\bar{u}, \bar{v})/180$, y por tanto que tengan el mismo signo es $(180 - d_a(\bar{u}, \bar{v}))/180 \geq (180 - d_1)/180$. De la misma forma, si $(\bar{u}, \bar{v}) \geq d_2$, para cualquier vector aleatorio la probabilidad de que tengan el mismo signo es $(180 - (\bar{u}, \bar{v}))/180 \leq (180 - d_2)/180$.

Conclusión. Perfecto! tenemos entonces nuestro esquema locally sensitive. El problema es que no sirve de mucho. Supongamos que queremos todos los pares de elementos con distancia angular de 20 o menos. La propiedad de arriba nos dice que los elementos asi de cercanos serán mapeados con el mismo signo con probabilidad $160/180 = 0{,}8\bar{3}$. Esto igual deja un 17 % de los elementos fuera, ¡es demasiado!

3. Amplificando familias de funciones sensitivas

Para magnificar las probabilidades, de forma que nos queden mas razonables, vamos a utilizar de forma paralela estas dos técnicas sobre una familia F de funciones sensitivas. Notemos que las funciones resultantes son funciones binarias y booleanas, solo nos van a servir para calcular si $f(x) = f(y)$ o $f(x) \neq f(y)$.

OR Tomamos b funciones en F, digamos $f_1, \ldots, f_b$, y nos quedamos con la disyunción de todas ellas. Más precisamente, la nueva familia de funciones F^{OR} contiene una función

[1]Vamos a suponer que no existe el caso donde $\bar{w}$ y $\bar{u}$ o $\bar{v}$ sean ortogonoales.

por cada conjunto de b funciones en F. Esa función f^{OR} se define como $f^{\text{OR}}(x,y) = 1$ si existe algún $f_i \in \{f_1, \ldots, f_b\}$ tal que $f_i(x) = f_i(y)$, y $f^{\text{OR}}(x,y) = 0$ en otro caso. Si la familia original era (d_1, d_2, p_1, p_2)-sensitiva, entonces la familia F^{OR} es $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitiva

AND Tomamos una conjunto r de funciones en F, digamos $f_1, \ldots, f_r$, y nos quedamos con la conjunción de todas ellas. Más precisamente, la nueva familia de funciones F^{AND} contiene una función por cada conjunto de r funciones en F. Esa función f^{AND} se define como $f^{\text{AND}}(x,y) = 1$ si para todo $f_i \in \{f_1, \ldots, f_r\}$ se tiene que $f_i(x) = f_i(y)$, y $f^{\text{AND}}(x,y) = 0$ en otro caso. Si la familia original era (d_1, d_2, p_1, p_2)-sensitiva, entonces la familia F^{AND} es (d_1, d_2, p_1^r, p_2^r)-sensitiva

Como vemos, tomar el AND hace disminuir las probabilidades en la definición de ser sensitiva, y el OR las hace aumentar. La gracia es ir intercalando estas construcciones para bajar p_2 y subir p_1.

Amplificando funciones para la distancia angular. Para nuestro ejemplo con distancia angular, el esquema es el siguiente.

Tomaremos b bandas o conjuntos de funciones, cada una con r funciones de LSH para distancia angular (dadas por r vectores distintos). Para dos vectores $\bar{u}$ y $\bar{v}$, vamos a decir que el LSH de $\bar{u}$ y $\bar{v}$ los asigna como *candidatos a ser similares*, que escribiremos como $LSH_{b,r}(\bar{u}, \bar{v}) = 1$, si para alguna de las b bandas se tiene que $f(\bar{u}) = f(\bar{v})$ en cada una de las r funciones.

Consideremos que la distnacia angular entre $\bar{u}$ y $\bar{v}$ es θ. Entonces la probabilidad que $LSH_{b,r}(\bar{u}, \bar{v}) = 1$ corresponde a $1 - (1 - ((180 - \theta)/180)^r)^b$.

Actividad Propuesta. Para distintos valores de b y r, experimenta graficando esta función. ¿Cuáles son buenos valores de b y r?

4. Bases de datos de vectores

Ahora usando funciones sensitivas podemos mapear un espacio de elementos a un espacio mas pequeño, además con la gracia de que los elementos mas cercanos en el espacio original quedan cerca en el nuevo espacio.

Una **Base de datos de vectores** es una estructura que usa este espacio más pequeño para recuperar rápidamente algunos de los elementos más cercanos a un elemento dado. Notar que decimos algunos, pues estas estructuras, al basarse en funciones sensitivas, tienen a ser probabilísticas: Con alta probabilidad entregan resultados que eran cercanos, y con baja probabilidad entregan resultados lejanos.

Vamos a ver una estructura que hace eso, llamada LSH Forest, aunque hay otras estructuras con propiedades similares en la literatura y la industria.

4.1. LSH Tree

Veamos primero como armamos un árbol.

Tenemos nuestro espacio métrico E y distancia d asociada. Además, contamos con una familia F de funciones (d_1, d_2, p_1, p_2)-sensitivas. Vamos a asumir que nuestras funciones son binarias: están definidas de E a $\{0, 1\}$. Nota que nuestro esquema LSH para distancia angular puede tomarse como una función binaria, haciendo $f(e) = 1$ cuando $e \cdot \bar{w}$ tiene signo positivo, y $f(e) = 0$ cuando $e \cdot \bar{w}$ tiene signo negativo.

Nuestro árbol funciona tomando ℓ functiones $f_1, \ldots, f_\ell \in F$. A cada elemento $e \in E$, le asociamos el bitvector $(f_1(e), \ldots, f_\ell(e))$.

En su escencia, un LSH Tree es simplemente un árbol de búsqueda sobre estos vectores: las hojas son cada uno de los strings posibles en $\{0, 1\}^\ell$, y apuntan a todos los elementos e tal que $(f_1(e), \ldots, f_\ell(e))$ es ese string. El nodo raíz tiene dos hijos, n_0 y n_1. El nodo n_0 es ancestro de todas las hojas correspondientes a strings que comienzan por el 0, mientras que el nodo n_1 es ancestro de todos los strings que comienzan por el prefijo 1. El resto del árbol se define inductivamente (el nodo n_0 tienen hijos n_{00} y n_{01}, cuyos descendientes son las hojas con prefijos 00 y 01, respectivamente, y así).

Para buscar los K elementos mas cercanos a un elemento e dado, procedemos a calcular su vector $(f_1(e), \ldots, f_\ell(e))$, y retornamos todos los elementos que compartan el mismo vector. Si aún no tenemos suficientes elementos, hacemos lo siguiente. Subimos un nivel, al nodo correspondiente al string $(f_1(e), \ldots, f_{\ell-1}(e))$, y buscamos todos los hijos ahí. Si aún no hay suficientes, vamos un nivel mas arriba, y retornamos los hijos del nodo correspondiente a $(f_1(e), \ldots, f_{\ell-2}(e))$, y así sucesivamente hasta completar al menos elementos. Muchas veces, cuando ya tenemos K elementos (o más), calculamos la distancia (usando d) de e con cada uno de esos elementos, y volvemos a ordenar de acuerdo a d.

Ejercicio. Para la distancia angular, un elemento e recibido como input y un elemento e' en E, y tal que $d(e, e') = \theta$, ¿cual es la probabilidad de que el LSH Tree retorne e' cuando recibe como input a e?

4.2. LSH Forest: Optimizaciones, más árboles

Por supuesto, en la vida real no es posible saber a ciencia cierta que contamos con cada uno de los elementos de E, si no solamente un subconjunto S de elementos iniciales.

Por otro lado, si guardamos todas las hojas necesitamos un árbol de tamaño, al menos, 2^k. Esto puede ser muy prohibitivo en la vida real. Entonces.

1. Nuestra estructura solo va a estar construida a partir de los elementos en S que conocemos. Lo hacemos top-down: calculamos primero $_1$, asignamos elementos a los hijos, y así sucesivamente.

2. Definimos un parámetro, llamémoslo H, que indica el numero mínimo de elementos asociados a una hoja. Si una hoja dada no tiene esa cantidad de elementos, entonces esa hoja se fusiona con su hoja hermana, y estos elementos ahora son asociados al padre de estas hojas.

3. Definimos otro parámetro, F, que indica la cantidad de árboles que tenemos al mismo tiempo. Tenemos ℓ funciones de LSH Hash por árbol. Ahora, cuando buscamos elementos cercanos a algun elemento e, retornamos todos los elementos que coinciden con el vector (o un prefijo) $(f_1(e), \ldots, f_\ell(e))$ *en cualquiera* de los árboles.